AFRL-IF-RS-TR-2002-265
Final Technical Report
October 2002

AGILITY: AGENT – ILITY ARCHITECTURE

Objecting Services and Consulting, Incorporated

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J357, G338

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-265 has been reviewed and is approved for publication.

APPROVED:

WAYNE BOSCO
Project Engineer

FOR THE DIRECTOR:

MICHAEL L. TALBERT, Maj., USAF
Technical Advisor, Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 2002	Final Jun 98 – Jun 02

4. TITLE AND SUBTITLE
AGILITY: AGENT – ILITY ARCHITECTURE

5. FUNDING NUMBERS
C - F30602-98-C-0159
PE - 63760E , 62301E
PR - AGEN
TA - T0
WU - 18

6. AUTHOR(S)
Craig Thompson, Tom Bannon, Steve Ford, Paul Pazandak, and Venu Vasudevan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Objecting Services and Consulting, Incorporated
6111 Baywood Ave
Baltimore Maryland 21290

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-265

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Wayne Bosco/IFTB/(315) 330-3578/ Wayne.Bosco@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
Today's agent systems are monolithic, centralized, and do not provide a clear migration path for integration with mainstream technologies (e.g., object and web technologies). The objective of the Agility project is to develop an open agent grid architecture populated with scalable, deployable, industrial strength agent grid components, targeting the theme "agents for the masses." The overall technical approach has been to deconstruct agent systems into components, then populate an open agent grid architecture with scalable light-weight agent grid components that are engineered to piggyback on existing and emerging standards (e.g., distributed objects, email, web, search engines, XML, Java, Jini). Three agent system components resulted from this work:

- a light-weight agent system that uses email for message transport (eGents),
- a constrained natural language interface system that can wrap agents and other Internet resources and operate over the web (AgentGram),
- a yellow pages service that uses Internet search engines to locate XML ads for agents and other Internet resources (WebTrader).

14. SUBJECT TERMS
Agent, Object, Internet, Web, Middleware, Scalable, Email, Natural Language Interface, Trader

15. NUMBER OF PAGES
78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Preface

We provide four top-level project summaries of the Agility project:

- the first follows in the form of a Final Report and summarizes our main accomplishments
- the second is in the form of a **final review presentation** and provides a .ppt description of our objectives and accomplishments. [*http://www.objs.com/agility/final/2001-10-25-CoABS-OBJS-Agility-Project-Review.ppt*]
- the third is our **public web page** [*http://www.objs.com/agility/index.html*], maintained throughout the project
- the fourth is our Government-only web page (password protected site, available upon request to qualified Government personnel) containing monthly, quarterly, and annual progress and financial reports and URLs of all technical presentations and reports.

Chapter 1

Overview and Summary of Agility Project

1.1 Problem

Today's agent systems are monolithic, centralized, and do not provide a clear migration path for integration with mainstream technologies (e.g., object and web technologies). As a consequence, though agent technology is identified as a promising high impact DoD software technology, it is not significantly impacting DoD, software technology or the mass market.

1.2 Objective

The objective of the Agility project is to develop an open agent grid architecture populated with scalable, deployable, industrial strength agent grid components, targeting the theme "agents for the masses."

1.3 Approach

Our overall technical approach has been to deconstruct agent systems into components, then populate an open agent grid architecture with scalable light-weight agent grid components that are engineered to *piggyback* on existing and emerging standards (e.g., distributed objects, email, web, search engines, XML, Java, Jini).

Specific objectives are to develop:

- a light-weight agent system that uses email for message transport (eGents),
- a constrained natural language interface system that can wrap agents and other Internet resources and operate over the web (AgentGram),
- a yellow pages service that uses Internet search engines to locate XML ads for agents and other Internet resources (WebTrader).

These components should operate standalone and/or interoperate with the CoABS grid.

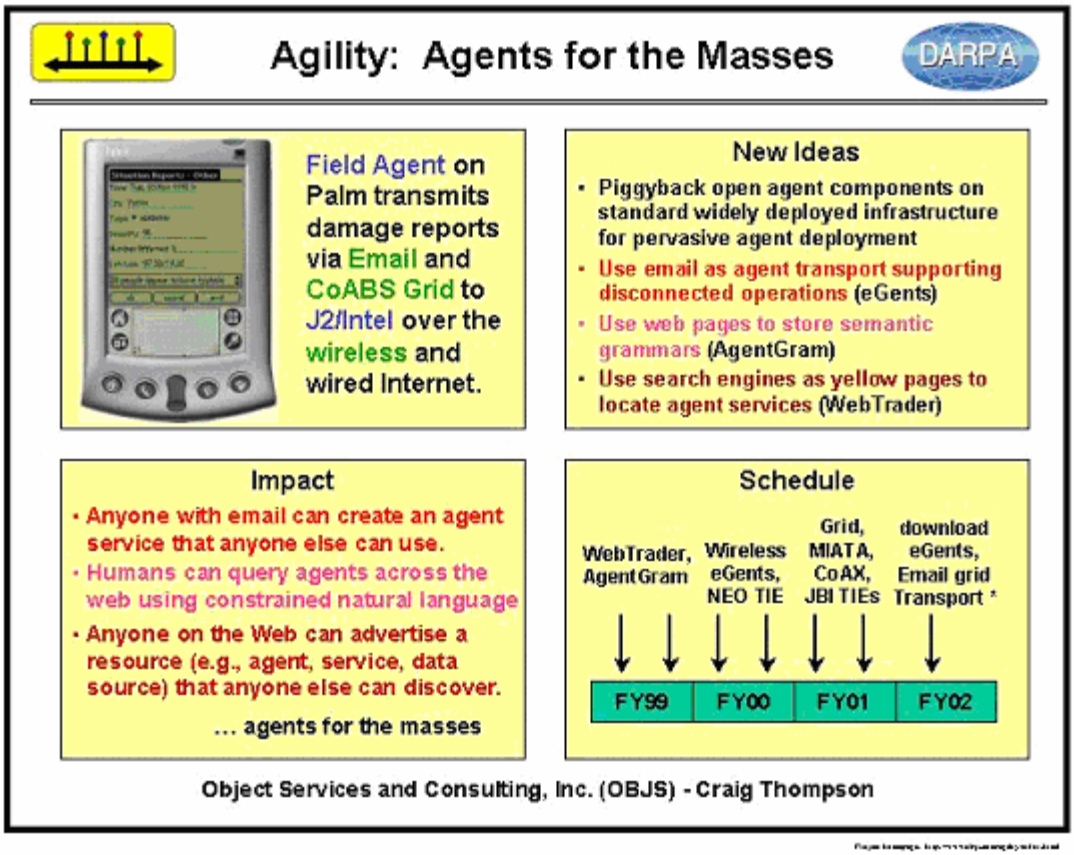


Figure 1: Agility Quad Chart Overview

1.4 Accomplishments

1.4.1 Architecture

In 1998 we presented **Strawman Agent Architecture** [*http://www.objs.com/agility/tech-reports/9808-agent-ref-arch-draft3.ppt*] to DARPA CoABS, ATAIS, and ALP, then submitted it to OMG Agent SIG and to FIPA. Parts of the presentation are now incorporated in the OMG Agent SIG **Green Paper on Agent Technology** [*http://www.objs.com/agent/agents_Green_Paper_v100.doc*], which we co-authored.

In 1999, our paper **Characterizing the Agent Grid** [*http://www.objs.com/agility/tech-reports/000304-characterizing-the-agent-grid.doc*] was submitted to and accepted for Jeff Bradshaw's book Handbook of Agent Systems. Sections of the paper appear in the architectural sections of the CoABS Grid Vision document (available on the GITI CoABS website, *http://coabs.globalinfotek.com/*, password protected).

1.4.2 Prototypes

To demonstrate our component-based approach, we are developed prototypes of three agent grid components (eGents, AgentGram, and WebTrader), described in Chapters 2, 3, and 4 respectively, and made the following technical progress:

- **eGents** are agents that communicate via email (see Chapter 2). We focused the bulk of our effort on eGents because of the widespread potential impact to DoD and industry. The potential impact of eGents is, anyone with email can create an agent service that anyone else can use. Imagine eGents attached to sensors, actuators, people, equipment, weapons, and locations as pervasive observers and actors. In the first years, we developed the basic eGent prototype which included the idea of encoding agent communication language in XML and demonstrated email used as an agent transport. We submitted both ideas to FIPA and they adopted both. More recently, we extended eGents to operate wirelessly on Palms and to support time-constrained publish-and-subscribe messaging consistent with the Joint Battlespace Infosphere (JBI) architecture. eGents should run on any Java platform; it was tested on NT/2000 machines; it connects to garden variety SMTP/POP3 email servers. A ported eGents runs on KVM (J2ME CLDC 1.0 FCS) on the Palm Vx.
- **AgentGram** wraps Internet resources including agents with menu-based natural language interfaces (see Chapter 3). It can operate over the web so the interface is remote from parser and resource. We designed a way to annotate web pages with grammars, a step towards the semantic web. We demonstrated AgentGram in the CoAX TIE where it was used for wildlife location queries at the last minute before a planned UN firestorm, forcing replanning. Potential impact: Humans can task and query agents using complex but understandable commands in constrained natural language that the system is guaranteed to understand. This technology can mix pervasively into all applications, both

on the desktop and the Web. The user interface client was implemented as a Java applet and also in Javascript, and tested with IE and Netscape browsers. The backend parser server is in C.

- **WebTrader** is a scalable robust trader component architected for the global grid (see Chapter 4). WebTrader locates advertisements, represented in XML stored on web pages, that have been indexed by industrial strength search engines. Advertisement types include agents, components, data sources, search engines, MBNLI grammars, other traders, channels, and other types of resources. Potential impact: In the spirit of the semantic web, anyone on the Web can advertise a resource (e.g., agent, service, data source) that anyone else can discover. WebTrader's user interface is a Java applet that runs in IE and Netscape; its server uses Apache web server and Thunderstone's Webinator 2.5 search engine, and can interface to other search engines.

We provide descriptions for these three technologies in Chapters 2, 3, and 4 respectively.

1.5 Technology Transition

1.5.1 Papers and Workshops

In addition to our architecture presentation and grid paper mentioned above, early versions of the three prototypes were described in a workshop paper:

- Agents for the Masses [*http://www.objs.com/agility/tech-reports/9902-agents-for-the-masses.doc*], Thompson, Bannon, Pazandak, and Vasudevan, invited paper, Agent 99 Workshop on Agent-Based High Performance Computing: Problem Solving Applications And Practical Deployment, Seattle, May 1 1999. [*http://www.cs.cf.ac.uk/User/O.F.Rana/agents99/index.html*]

Also, we were/are on the program committees for:

- Infrastructure for Scalable MultiAgent Systems, workshop at Autonomous Agents June 3-4 2000 Barcelona [*http://www.cs.cf.ac.uk/User/O.F.Rana/agents2000/*]
- OMG Grid Workshop, July 2001 [*http://www.omg.org/news/meetings/tc/software_services_grid_workshop.htm*]
- Agent based Cluster and Grid Computing, Berlin, May 21-24 2002 [*http://www.cs.cf.ac.uk/User/O.F.Rana/agent-grid-2002/*]

1.5.2 Software and the Grid

All three prototypes were extended to operate over the over the CoABS **24x7 Grid** [*http://agent1.globalinfotek.com/*] and were available for a year on the grid (see the grid archives and individual descriptions of eGents, AgentGram, and WebTrader).

1.5.3 Technology Integration Experiments (TIEs)

We participated in four TIE efforts covering DoD domains:

- **NEO TIE** - In the non-combatant evacuation order domain, AgentGram was used to request the location of evacuees. WebTrader was used to locate various agent services. This was demoed at several CoABS meetings and as part of the NEO TIE demonstration now running at AFRL. Developed later, eGents was used to monitor and observe health and location status of non-combatant evacuees in a NEO-like scenario.
- **MIATA TIE** - In the disaster recovery domain, eGents sent field reports (e.g., bridge out, typhoid outbreak). This was demoed at CoABS Workshop in Miami using a wireless Palm.
- **CoAX TIE** - In the coalition domain [*http://www.aiai.ed.ac.uk/project/coax/demo/2001/*], eGents sent biosurveillance reports (e.g., location of elephants threatened by a planned UN firestorm in Safari Park Binni Wildlife vignette). AgentGram was used to query for the location of elephants near a UN firestorm. This was demoed at the CoABS Workshops in Miami and Nashua. See **CoAX TIE avi** (.exe includes TechSmith TSCC Codec and viewer - 4.2MB - *http://www.objs.com/agility/tech-reports/0107-CoABS-Agility-Nashua/OBJS-CoAX-TIE-1024-768-256-TSCC.avi.exe*). Our work on the CoAX TIE is included in the paper:
  - Allsopp, D.N., Beautement, P., Bradshaw, J.M., Durfee, E.H., Kirton, M., Knoblock, C.A., Suri, N., Tate, A. and Thompson, C.W. (2002) "Coalition Agents Experiment: Multi-Agent Co-operation in an International Coalition Setting", Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations (KSCO-2002), Toulouse, France, 23-24 April 2002. [*http://www.aiai.ed.ac.uk/project/coax/doc/2002-ksco-coax.doc*]. Revised for KSCO special issue of *IEEE Intelligent Systems*, forthcoming.
- **JBI TIE** - In the small unit operations domain, commanders drill down via a map to subscribe to and monitor troop and platoon status during an attack and fuselets notice patterns (e.g., chemical attack, soldier wounded). This was demoed at CoABS Workshop in Nashua. See **JBI TIE avi** (.exe includes TechSmith TSCC Codec and viewer - 2.9MB - 2:14 min - *http://www.objs.com/agility/tech-reports/0107-CoABS-Agility-Nashua/OBJS-CoAX-TIE-1024-768-256-TSCC.avi.exe*).

1.5.4 Inventions

We completed two patent applications:

- *Network Query and Matching System and Method* - U.S. Patent App. No. 09/407,555
- *Guided Natural Language Interface System and Method* - U.S. Patent App. No. 09/634,108

1.5.5 Related DARPA Programs and Other Contracts

Frank Manola completed the **ALP-CoABS Integration Final Report** [*http://www.objs.com/agility/tech-reports/9907-CoABS-ALP-Integration-Final-Report.html*] in July 1999. Program managers Jim Hendler and Todd Carrico indicated that we did a good job and both felt more integration work might occur at a later date but neither was ready to fund more work at that time on CoABS-ALP TIEs.

Work begun on eGents under CoABS has led to two additional contracts:

- the DARPA UltraLog [*http://www.darpa.mil/tto/*] MsgLog contract (in progress) - the idea is to provide the ALP-Cougaar agent based system with multiple communication transports (email and NNTP in addition to RMI) and then use policy management to select among them to insure survivable, robust message delivery in chaotic environments. David Wells described our Msg*Log project, including its origins in the CoABS Agility project, in Msg*Log: Email-based Agent Messaging to Improve Robustness in a Distributed Logistics Planner [*http://www.objs.com/agility/tech-reports/20010501-STC.doc*], a paper for the Software Technology Conference (STC 2001), Salt Lake City, Apr 29-May 4, 2001.
- an SBIR II with ScenPro as prime (to begin FY02) - the idea is to extend eGents in several ways to support small unit operations

1.5.6 Standards

We influenced agent standards:

- FIPA [*http://www.fipa.org/*]
  - We submitted eGents: Agents over Computational E-mail [*http://www.objs.com/agility/tech-reports/9812-FIPA-Comp-Email-Agents.html*] to FIPA CFP-99. The document described two candidate standards: a method of encoding FIPA ACL in XML and an API for transporting FIPA ACL via email. FIPA subsequently adopted standards in both areas.
  - We submitted Strawman Agent Reference Architecture [*http://www.objs.com/agility/tech-reports/9808-agent-ref-arch-draft3.ppt*] in response to FIPA CFP-99 at the request of the Sun Agent Research Team.
- OMG Agent SIG [*http://www.objs.com/isig/agents.html*]
  - We organized and co-chair OMG Agent Working Group, maintain its web page including agendas and minutes of most meetings, and have used this forum to showcase DARPA work on agents, grid, ontology, and mobility to mutual benefit.
  - OMG Document: Mission Statement, Craig Thompson, March 23, 1999
  - OMG Document: OMG-FIPA Liaison, Craig Thompson, March 24, 1999 - passed by vote in OMG, passed by vote at FIPA Nice
  - OMG Document: Agent Technology RFI, Craig Thompson, March 23, 1999 - issued to industry, responses are here.

- OMG Document: Agent Grid Presentation, Craig Thompson, August 23, 1999
- OMG Document: Agent Glossary, Craig Thompson, September 10, 1999
- Tech Note: Requirements for an Agent Discovery Service, January 2000
- OMG Document: Agent Technology Green Paper, August 2000 - added sections on agent architectures, grid and ilities, and on the Relationship of Agents and Objects, pp 41-45.
- OMG Document: White Paper and Roadmap for Agent Technology Standards (rev .04), March 2000

1.5.7 Commercialization

We are working on productizing AgentGram via the spinout LingoLogic.com [*http://www.lingologic.com/*].

1.6 Future Directions

Some of the more important future directions are:

- Extend eGents for automated deployment to new platforms, download eGent apps to the field as situations change, and improve eGents security - currently eGents requires manual installation limiting fast fanout of new eGent applications.
- Extend the CoABS grid to support survivable multi-transport messaging - leverages our ongoing Cougaar Ultralog survivable, policy-driven messaging work, which got its start with eGents! This is a step towards demonstrating how to control system-wide properties (QoS ilities) in agent/grid architectures.
- Rendezvous selected components with DAML.

Additional next steps for eGents, WebTrader, and AgentGram are described in the Final Review [*http://www.objs.com/agility/final/2001-10-25-CoABS-OBJS-Agility-Project-Review.ppt*].

1.7 Conclusions

We started with a reasonable thesis and were able to demonstrate feasibility of the approach of deconstructing agent systems into components and rebuilding industrial strength components, piggybacking their implementations on already pervasive technology. This does appear to be a viable route for getting agent technology into the mass market in a way that provides a migration path.

Chapter 2

eGents Prototype: Agents communicating via Email

2.1 Objective

The objective of the eGents project is to develop a modular light weight agent system that uses email as a transport layer. There are a number of advantages to using email: it is pervasive and industrial strength, and it already supports disconnected operations, message queuing, message filtering, firewall permeability, logging, and security. So an email-based agent system should not have to replicate this functionality. Instead, anyone with email can create an agent service that anyone else can use.

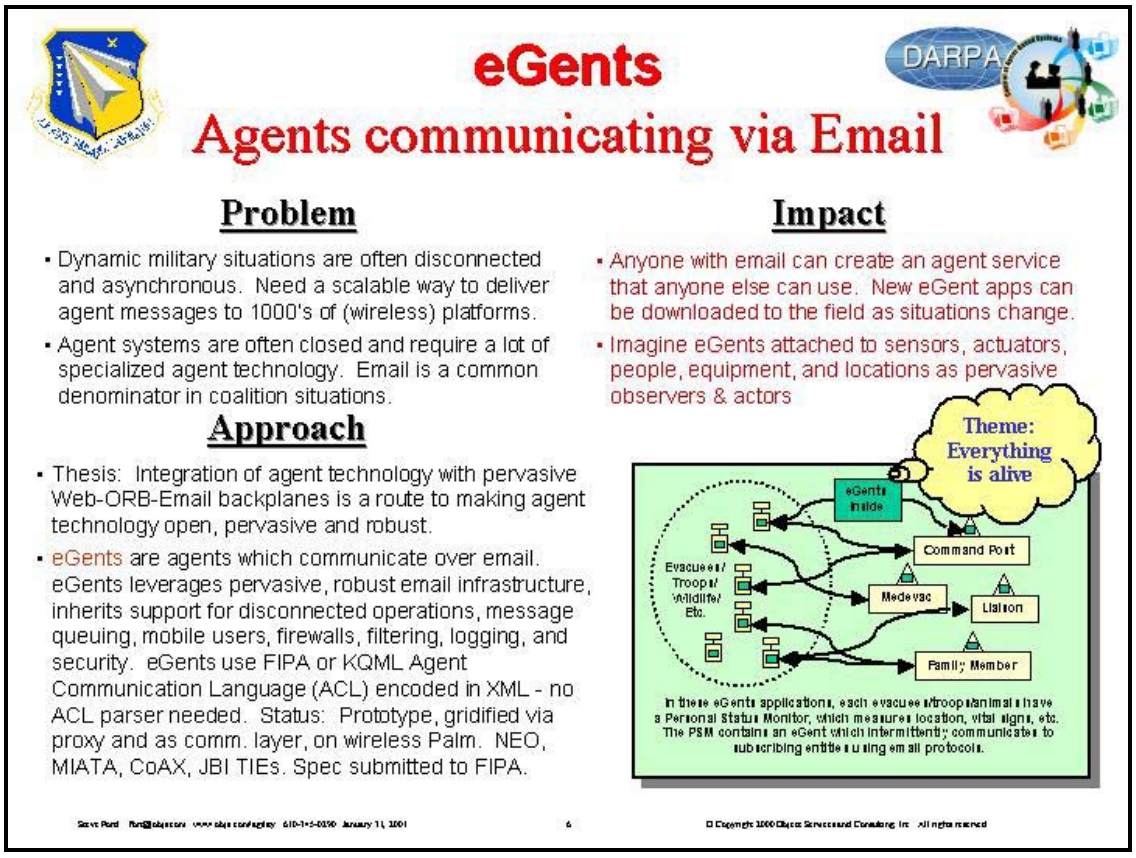


Figure 2: Overview of eGents

Specific scientific and engineering subgoals were:

- develop a lightweight agent system that uses email-based transport
- support (as subset of) FIPA agent communication language (ACL) represented in XML
- demonstrate eGents running on PDAs (e.g., Palms, any device running KVM)
- demonstrate eGents in DoD scenarios

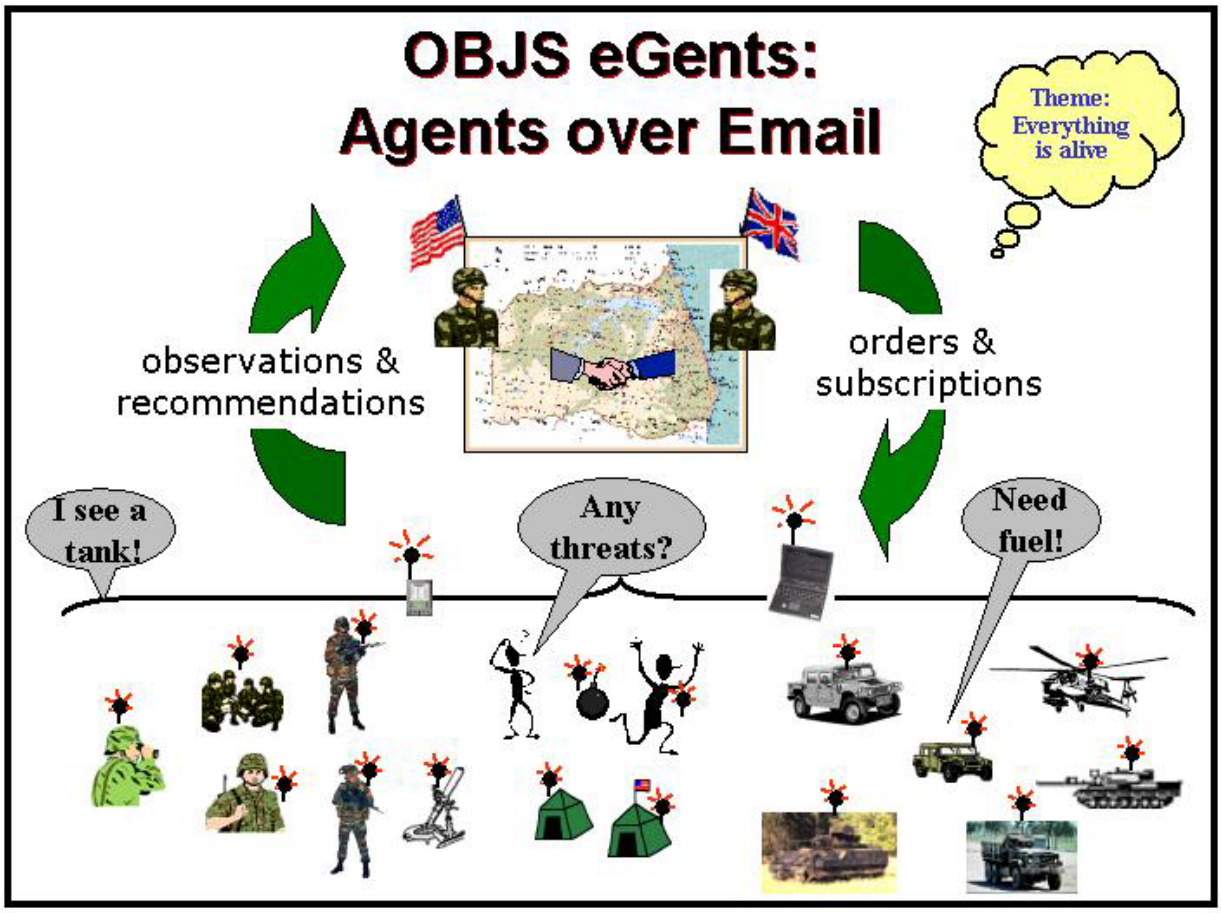


Figure 3: "Everything is alive" if it can sense, process, and communicate

2.2 Technical Accomplishments

2.2.1 Motivation

Despite the spread of web and distributed object technology, enterprise applications remain hard to build. CORBA and other distributed-object based solutions require heavyweight investments of software on the desktop, and a fair amount of investment in the glue software. Web and e-mail are lightweight and pervasive, but currently used for document transport, not distributed computing. The goal in eGents is to adapt these transports to support distributed computations (specifically distributed agent computations), thus providing an agent platform that is useful, lightweight and leverages currently deployed technologies. In particular, eGents borrows the notion of agent communication languages as a messaging language from the agent world, and integrates it with email as a computational transport, XML as the enterprise EDI language, and Java as the computational engine at the desktops. To test our ideas, we focused on applications which benefit from eGents leverage of the vast and pervasive email infrastructure, especially those needing support for asynchronous and disconnected operations, message queueing, firewall permeability, filtering, logging, and security.

2.2.2 Architecture

The eGents framework is comprised of two types of agents: *platform agents* and *task agents*. Task agents are distributed java components that are wrapped using an agent wrapper framework so as to send, receive and respond to ACL messages. As their name indicates, they perform a particular task on a specific machine. Platform agents manage task agents, and mediate access to them. Task agents on a single machine are controlled by a per-machine platform agent which launches and terminates task agents (the task agents being threads executing as part of the platform agent process). Agent applications consist of communicating task agents that are dispersed across different machines (perhaps different administrative domains) across the WAN.

eGent task agents dispersed across a WAN communicate via a bi-level ACL bus using (a) an email-based ACL bus between platform agents and (b) the JavaBeans protocol between a platform agent and a task agent or between two colocated task agents. ACL messages are addressed to a named task agent belonging to a specific platform agent, and each platform agent has a unique email address. The eGents framework converts the ACL performative into an XML document and emails it to the destination platform agent. The destination platform agent parses the XML-based ACL message into a Java object, and forwards it to a local task agent using JavaBeans as a messaging protocol. Javabeans supports the *observer* design pattern whereby objects can *subscribe* to change events on other objects. Task agents use this to subscribe to changes in an ACL-mailbox object into which the platform agent stuffs incoming ACL messages.

The eGents ACL bus reuses standard Java, XML and email technology to build an ACL bus that has previously required proprietary KQML/FIPA ACL parsers and proprietary TCP/IP based transports. The use of XML as the EDI transport makes use of widely available XML parsers to parse the content at the receiving end, avoiding the engineering problems reported in some ACL-based agent efforts. JavaBeans provides similar benefits within a single machine.

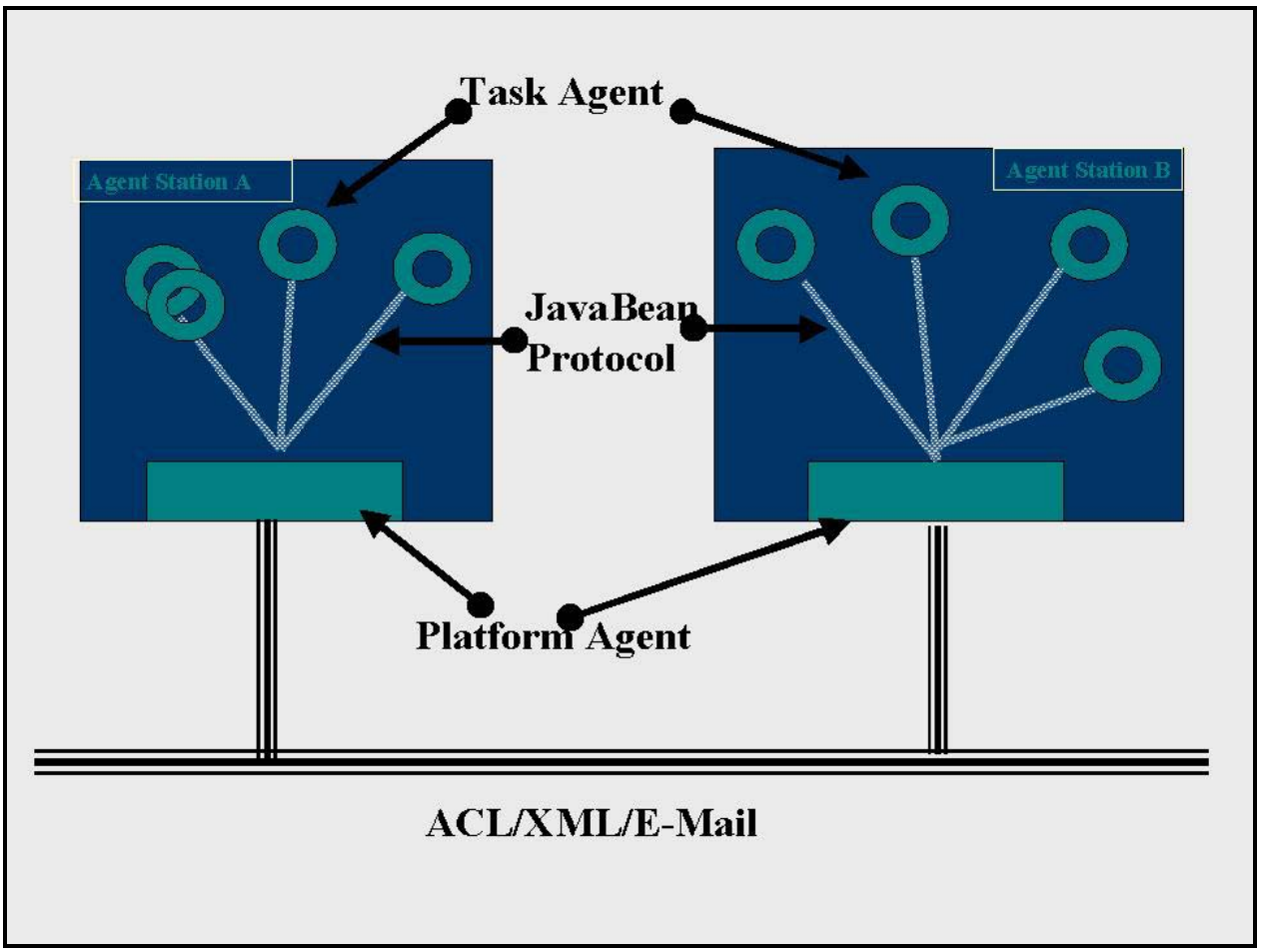


Figure 4: eGents Architecture

The agent wrapper framework uses a simple *microscripting* language to map ACL messages into sequences of method invocations on the Java component that implements the agent. One microscript statement corresponds to one task agent and one kind of performative. Blocks of microscript statements collectively specify how a particular agent type would respond to all ACL performatives that it handles. A per-machine *agentmap* (file) maintains the microscripts for all task agent types that are supported by a platform agent. For instance, the microscript statement below says that:

- the Glimpser agent maps to the com.objs.tao.Glimpser component

- the FIPA *subscribe* performative is implemented by calling the setWatchWord and setDistance methods (in that order)
- maps parameters in the *subscribe* performative to method arguments

```
Glimpser com.objs.tao.Glimpser subscribe #setWatchWord 1
#setDistance 2 #changes
```

The microscripting language emphasizes simplicity and minimality over full function workflow capabilities. As we gain experience with component wrapping, we will add to the language or consider integrating with other workflow specification mechanisms.

Agent wrappers use Java's dynamic class loading and reflection capabilities to execute the performative implementation based on a microscript specification. The microscript interpreter parses the microscript, dynamically loads the associated java class object, and uses reflection to dynamically compose the method invocation string and to execute methods on this class object. Script-based wrappers offer the potential advantage that agent implementations can be dynamically changed in a running agent computation.

2.2.3 Initial applications

2.2.3.1 Technical Report Dissemination

As researchers, we spend a lot of time finding and cataloging technical papers that are relevant to our research, and disseminating this information to colleagues in our informal network. The forwarding of emails is a manual process that is tedious and error-prone for that reason. An alternative would be to establish lightweight repositories on everybody's desktop (Unix directories indexed by Glimpse servers) and to support subscription based access to these repositories using personal agents. That is what our first eGent application demonstrates, Technical Report Dissemination.

In this application, a publication repository is modelled as a single UNIX directory on one OBJS machine. This directory is indexed by a Glimpse server, making the index information accessible to the eGents platform running on this machine. Other OBJS users can subscribe to this repository by commanding the local eGents platform to launch personal monitoring agents (PM agent for short) on their behalf. These PM agents continually monitor the publication repository for newly deposited publications of interest to the owner of the personal agent. When new publications are discovered, the remote user (owner of the PM agent) can launch packager agents on the repository machine, which package the new publications into a multipart email and dispatch it to the subscribed user.

This application was chosen as an initial eGents scenario because it is accepted as a useful application and has at least a couple of well known non-agent solutions, notable Dumais' latent semantic indexing based system and SIFT from Stanford. It was felt that this application could

serve not only as a demonstration of eGents, but in the longer term as a benchmark for comparing and contrasting agent and non-agent solutions.

After completing the Technical Report Dissemination application, we ported eGent from Unix to NT. In the eGent demo application, we replaced the Glimpse dependence with the DOS Find utility.

2.2.3.2 Personal Status Monitor

In July 1990, we packaged the eGent project (code, license, documentation) and sent the system to Ed Killian at Rome who installed it successfully and commented "everything worked." Killian sent some hints for improving the install and test documentation and commented "It is cumbersome to type or cut/paste/modify the queries. It would be nice to have some kind of better interface for this." This led us to work on an improved demo for eGents (and some new kernel capabilities). The new demo shows evacuees who have a Personal Status Monitor that has *eGents inside* that communicates periodically (ACL sent via email) to subscribers (command post, medevac unit, state department monitor, ...). We demonstrated the PSM application at the CoABS Science Fair (October 1999). A number of attendees stopped to see our three demos including Jane Alexander (Deputy Director of DARPA). New ideas to follow up include: beeper for eGents (Hendler), USMTF messaging for MBNLI (Alexander), and MBNLI interfaces to Intelink databases (Hendler).

2.2.4 Advantages and Lessons Learned

Some of the real (and potential) lessons learned from this prototype were:

- ACLs make a nice messaging language. The fact that they are human readable makes the system easier to debug. Having said that, there are no miracles to ACLs beyond being a next-generation messaging language with some higher-level primitives (e.g multi-party negotiation).
- The fact that messages are human readable, human generatable and sent over e-mail makes it possible to substitute humans for agents and vice-versa. Agent A sending an ACL-over-XML-over-email to agent B does not know (or care) whether B is a program or a person. This allows applications to be prototyped with the human-in-the-loop coordinating task agents, and then subsequently replace the human with an agent.
- Building a logger is easy, just bcc: all ACL messages to a logger email address.
- Easy reconfiguration of the human/agent mesh that makes up the application requires a mediation architecture. One where agent-agent communication is actually agent-mediator-agent. Changing the application logic is then reprogramming the mediator.
- The benefit of agents so far is that of using ACLs. Any speculative advantages in terms of component discovery, automated assembly of the application, ability of personal agents to tune the information feed based on user feedback are all possible but not demonstrated. The idea is that this would be a lightweight and totally pervasive (and scalable) platform to try out such experiments.

2.2.5 Palm implementation of eGents

As mentioned, at the CoABS Science Fair (October 1999), Jim Hendler (DARPA CoABS Program Manager) suggested that we extend eGents to operate nomadically, perhaps using a pager. We subsequently determined that pagers at that time were not an ideal platform for eGents because of the near lack of programmability and the limited size of pager messages. It is not hard to get trivial eGents messages to the pager, its just that nothing interesting can execute there with then-current pager technology so it was at best a trivial use of eGents. But the Palm Pilot PDA proved to be a good candidate so we chose this path.

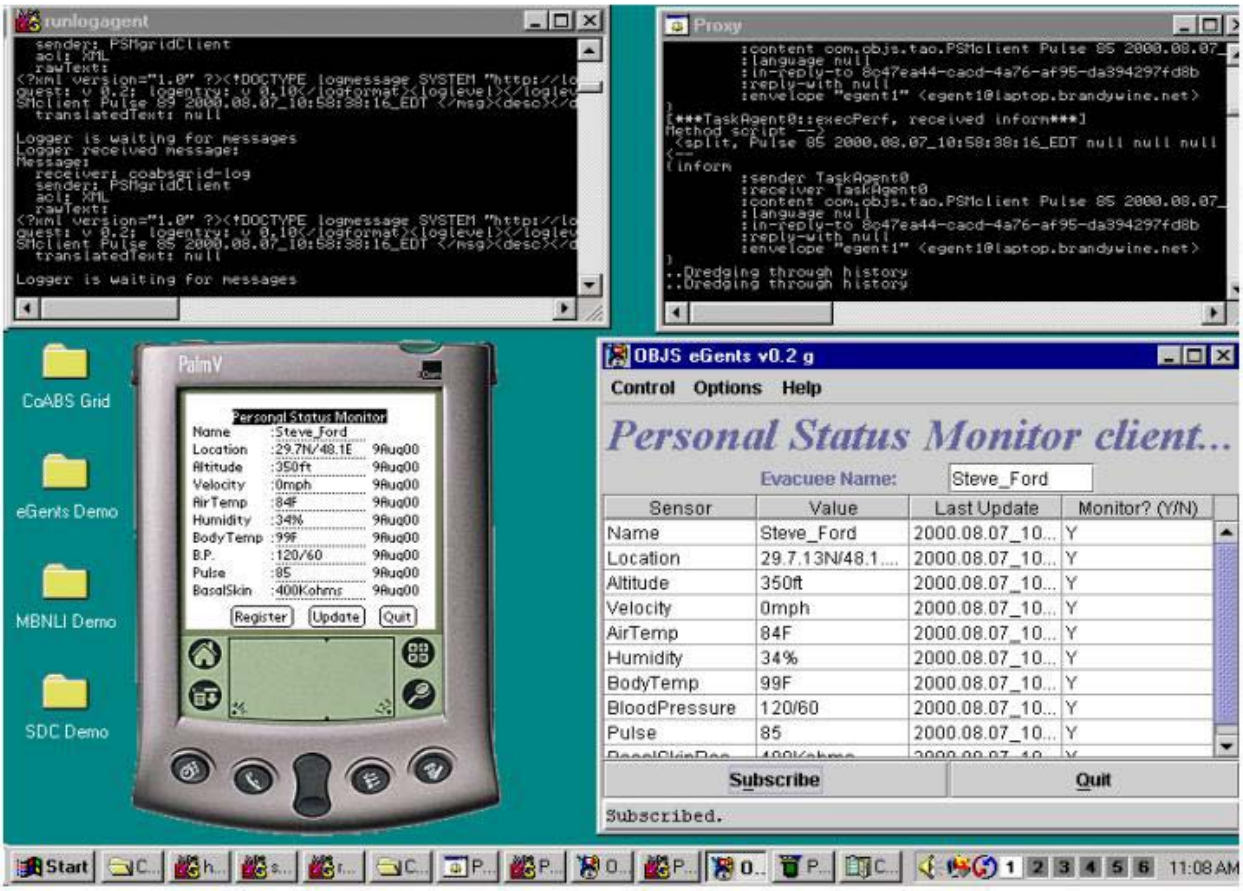


Figure 5: Personal Status Monitor demonstration

The hardware platform we settled on consisted of Palm Vx, Novatel Minstel V Wireless Modem, Omnisky Wireless Internet Access, and AT&T Wireless Cellular Service. The software was more the problem. Normal Palm applications are developed in C++. We also looked at Sun's K Virtual Machine (KVM - *http://java.sun.com/products/kvm/*), a then new Java runtime environment that was built from the ground up to make an extremely lean implementation of the Java virtual machine for use in devices that have a small memory footprint, like digital cellular

phones, pagers, mainstream personal digital assistants, low-end analog set-top boxes, and small retail payment terminals. We chose the latter as our purpose was to show that agent technology is not standalone and monolithic and therefore irrelevant to mainstream computing technology, but rather that it can be and is being integrated with mainstream technologies.

The biggest time sink was figuring out how to do email programmatically from a Java application on the Palm. We could not locate SMTP/POP3 class libraries that run under KVM (JVM for the Palm) nor even equivalent C++ email libraries. Palm Java (circa January 2000) was very immature, buggy, slow and a resource hog, but it worked. Mostly, people were doing graphical things with it. The most common area for problems was network access. Our plan was to use Multimail. Unfortunately, this idea wouldn't work, as KVM apps cannot access Palm apps, because KVM doesn't support JNI (Java Native Interface). KVM is a very restricted Java for use on small devices.

We decided to implement POP3 and SMTP for the Palm in KVM Java. We found a simple implementation of POP3 and started to port it to KVM -- not easy. So much of Java is missing that there's hardly a line of code you'd normally write that can go unchanged plus the codebase is not very stable and the debugging tools are poor. We ported the eGent code (sans email support) to KJava (Java for handhelds), stripping out all unnecessary classes, added others, integrated a KVM-compliant parser into the code (replacing the IBM parser), ported the demo Personal Status Monitor (PSM) and PSM client codebase and interfaces to KVM. We completed implementing POP3 for PalmOS in Java (KVM) and a test app, integrated it with the port of eGents, and ported it again to J2ME CLDC 1.0 Beta (the new version of KVM), and then ported a sufficient subset of JavaMail to KVM to support eGents' SMTP needs. Fortunately and unfortunately, PalmOS version 3.5 and CLDC 1.0 Beta came out in the middle of this. It was fortunate because the size of the Java heap increased by 4x, and without that, it might have proven impossible to run egents. Unfortunately, because it introduced some new different bugs, changed the network API, and temporarily invalidated our wireless access to the Internet. In 2Q00, we integrated all components into the eGents platform under J2ME CLDC 1.0 Beta (PalmOS) including the Personal Status Monitor (PSM) demo, and tested and debugged . We got it working on kvm.exe, and with more work on the POSE (Palm emulator) and the Palm itself but it was too slow.

Performance analysis pointed to POP3 message reception as the biggest problem, about 4X slower than SMTP message transmission. Focusing on that, we were able to speed message reception up to the level of transmission. It is now fast emough for the kind of applications we are targetting but slow for demo purposes. To run the demo we ran at the Science Fair takes about *three minutes* from the point that one clicks "subscribe" on the client until the client's fields are updated on its display. That involves one message send from the client and received by the server, and then one response sent by the server and received by the client. The messages were sent via the mail server on internal.objs.com. Most of the time is spent sending and receiving the messages, although xml parsing seems unnecessarily slow too, and some time is wasted in our primitive thread management. This all can be sped up quite a bit using standard techniques.

At this point, we turned our attention to gridifying eGents.

2.2.6 Gridifying eGents

CoABS Grid. The CoABS grid is a JINI-based implementation of an agent interoperability platform developed by GITI, the DARPA CoABS program integration contractor. It is an important, on-going experiment in agent system interoperability. As described elsewhere, we contributed architectural ideas to the grid. But in addition, we developed three standalone agent components (eGents, WebTrader, and AgentGram) that can play a role as grid components or services. As part of the Agility eGents project, we worked on two approaches to connecting eGents to the CoABS Grid described below. *At the same time, we note: eGents taken alone but piggybacked pervasively on Email is arguably a very different but also very powerful basis for an agent grid.*

eGent-Grid Proxy, 7x24 grid and eGents launch page. In July 2000 we gridified eGents and the Personal Status Monitor (PSM) demo using an eGent-grid proxy approach - see the two figures below. We can send messages from normal client GridAgents to GridAgent/eGent proxies (a Grid Agent that is also an eGent), which then translates the incoming message, which is in KQML, into an eGent message in FIPA ACL (pretty close alignment of fields, actually), and then forwards it to an eGent. To do so, we implemented the eGentGridAgent class, which is a generic proxy for registering eGents on the CoABS Grid, and relaying messages between Grid Agents and eGents. We modified the Java and KVM PSM Server eGents to (optionally) register with the Grid via an eGentGridAgent. We built a Java PSM Client Grid Agent for end-users to use to subscribe to PSM Servers, either eGents or Grid Agents. We demonstrated this new capability at the CoABS Workshop in Boston (August 2000). Thereafter we prepared eGents and the PSM demo for permanent installation on the 7x24 Grid [*http://agent1.globalinfotek.com/status.html*], and maintained it for most of a year (see Personal_Status_Monitor in the grid archives). This required some changes to bring the demo up dependably and usably on the 7x24 Grid and also to enable multiple clients' subscriptions to a single server. The latter involved making eGents handle connects and disconnects from multiple simultaneous clients reliably and intuitively. The public demo worked for multiple servers and clients, both Grid clients and eGents clients, operating simultaneously. We also provided a launch page with documentation for using and downloading the PSM demo.

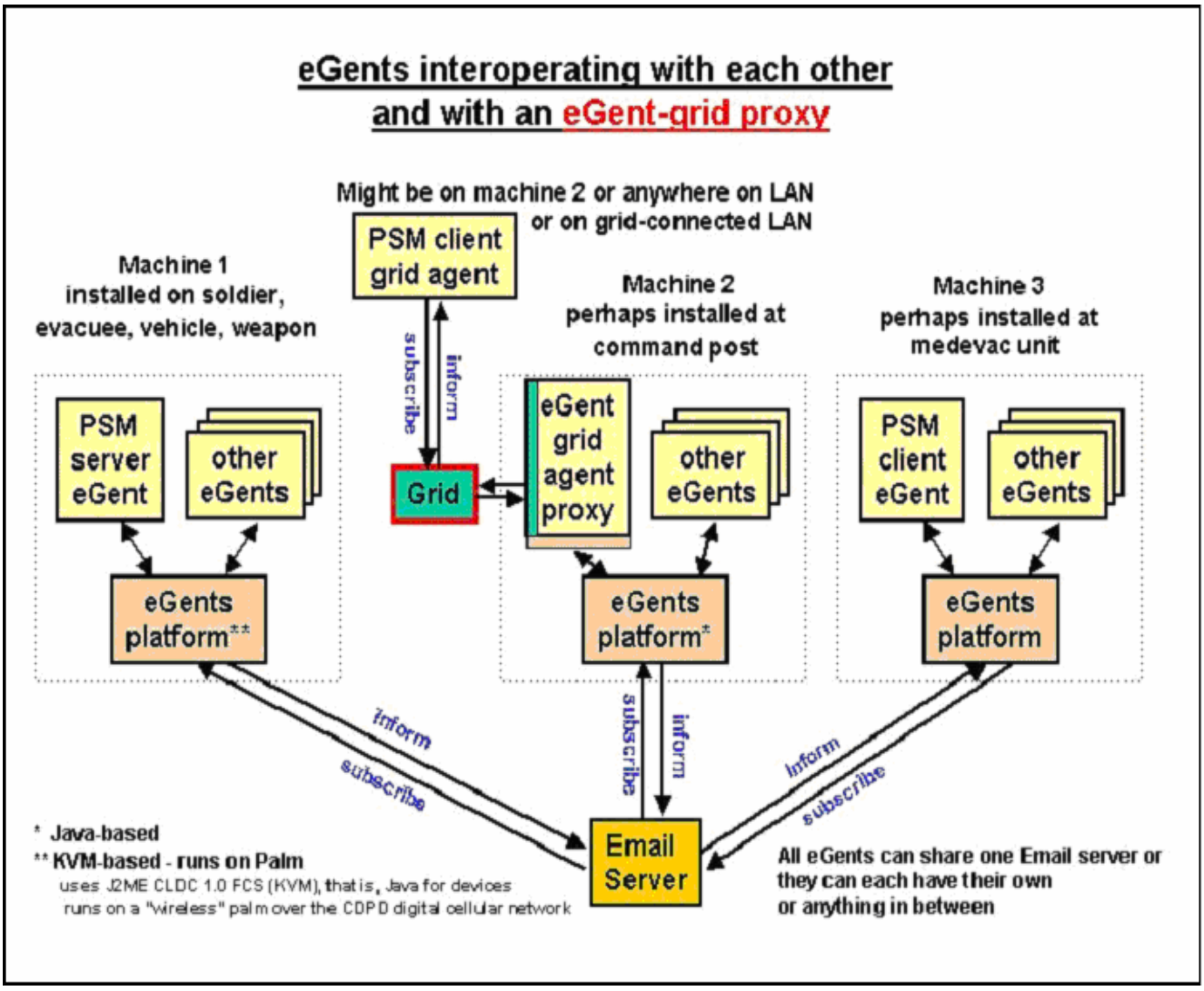


Figure 6: eGents can communicate via the CoABS Grid using an eGent grid agent proxy

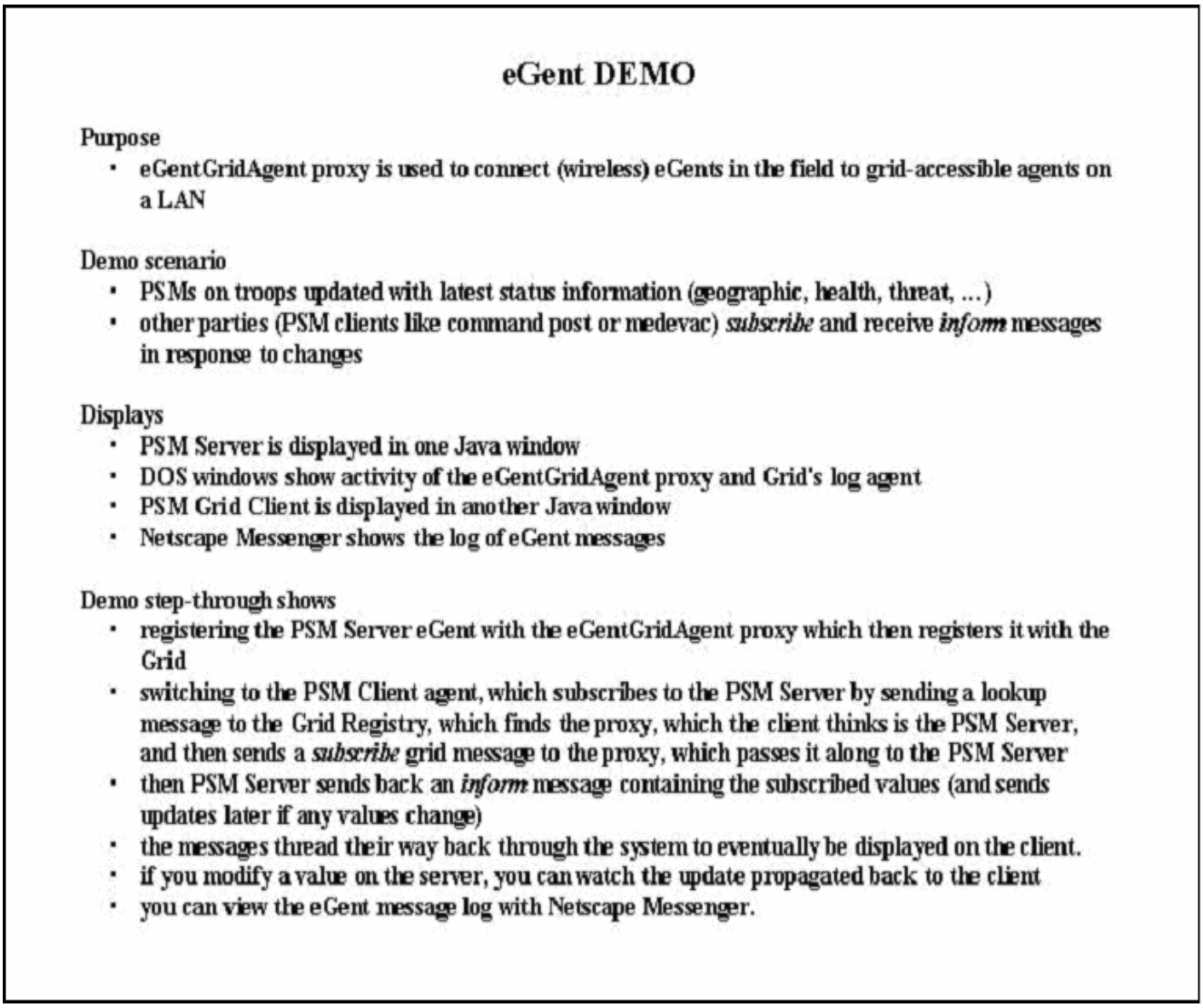


Figure 7: Script of eGents using the Grid

Towards an Email-based Grid Transport. In January 2001, we began a short-lived experimented with another way to connect eGents to the grid, this time using eGents (actually just email) as a grid com layer. This time, we wanted to consider how we could modify the grid to use an Email-based transport. We made good initial progress towards this goal. The initial approach was at the Java level (that is, below the grid and JINI). We looked at the RMI source and discovered that RMI allows one to substitute its socket-based transport layer with sockets of your own design, such as sockets that support SSL (encryption) or eGent-based sockets, so that is the path we took. We created an Email Sockets library and successfully used it as the RMI socket factory in a test RMI program such that all the RMI traffic including RMI registry lookups and distributed garbage collection (DGC) activity was routed over email instead of the usual sockets. We chose Unix-style sockets as one API for this email-based messaging capability because sockets are the predominant interprocess (IP) communications mechanism in many OSes including Windows and Unix and thus make a good direct or indirect building block, and because you can slip in custom sockets to handle all RMI traffic in Java with one line of code! We identified five integration points that Email Sockets could be used in or with a Java program in order to modify the program to use email for IP communication. We got an RMI test progam (a client-server pair) running in time for Miami. We wrote a second RMI program ("CoABS Chat") to be more demo-friendly. We also created a pared-down version of the CoABS Grid demo program "June Demo" to use as the starting point to inject Email Sockets into the Grid. We got the program to run successfully on the machine standalone, as well as using the 7x24 GITI Grid machine. We began to investigate the relationship of Jini to RMI (Jini uses RMI) and to determine how Email Sockets will play in the Jini-based Grid. In February, we decided we had to discontinue this effort in FY01 due to the funding shortfall and higher priority TIEs. In April we were able to transfer some of our technical effort on this subproject to our Msg*Log contract, which is part of the DARPA Ultra*Log program. There, we are focusing on adding eMail and NNTP as transports to ALP/Cougaar agents and eventually will focus on creating an adaptive, survivable message transport capability. So our early results, conceived and begun under Agility, will bear fruit in another grid-like distributed architecture, fulfilling this subproject's research objective.

2.3 Technology Transition

In December 1998, we responded to the 5th FIPA [*http://www.fipa.org/*] Call for Proposals on agent technologies suitable for standardization with a proposal for **FIPA E-Gents: Agents over Computational E-mail** [*http://www.objs.com/agility/tech-reports/9812-FIPA-Comp-Email-Agents.html*] which described certain aspects of eGents. In that paper, we identified two potential component standards: one for encoding FIPA ACL in XML so there is no need for a separate ACL parser; and the other for an API which allows email agents to send and receive ACL messages. FIPA has since accepted both proposals. We described and/or demonstrated eGents at CoABS Workshops at Northhampton (June 1999), Science Fair in Arlington (October 1999), Atlanta (February 2000), Boston (Augut 2000), Miami (January 2001) and Nashua (July 2001). In addition to standalone demos, eGents was used in three DoD-oriented technology integration experiments (TIEs) described below.

2.3.1 MIATA TIE

In the disaster recovery domain (MIATA TIE), Palm-based eGents was used as a Situation Reporter in the simulated aftermath of Hurricane Mitch (see figure below). The scenario is that human observers scattered around the Honduras theatre are equipped with wireless Palms, and they submit "Situation Reports" (essentially filling out eGents-based forms) to J2 (INTEL) when they come across situations of interest. There are several different kinds of reports supported (i.e. city, bridge, road, weather, other). The demo focussed on "other reports", of which several types are supported (flood, mudslide, epidemic, fire, evacuation). The situation actually reported in the demo was an evacuation report saying that the President of Honduras was stranded away from the capital. The J2 passed it to the JTF Commander, who gave the order to assign two helicopters to pick him up. This actually happened during Hurricane Mitch. For the demo, we worked with Mark Burstein (BBN, MIATA coordinator) and David Diller (BBN, MIATA scenario using eGents). In the demo, Palm-based eGents sends situation data to a Maple Sim GridAgent. The Palm itself was tethered via a serial cable (as unassisted wireless solutions did not work dependably in a conference room setting) to the laptop, where the mail server, Grid, and Proxy to Maple Sim reside. We used kAWT this time, which raises the level of Palm graphics programming a notch. Much of the work was in fitting the demo into memory and speeding it up. We had to toss out significant parts of eGents to make the demo fit. This was demoed at the CoABS Workshop in Miami (January 2002). Following the demo, at David Diller's (BBN) request, we packaged and delivered the MIATA Field Agent demo to BBN for incorporation in a permanent and/or traveling version of the MIATA demo. In the process, we installed a public domain mail server (Apache James) and got eGents to work with it.

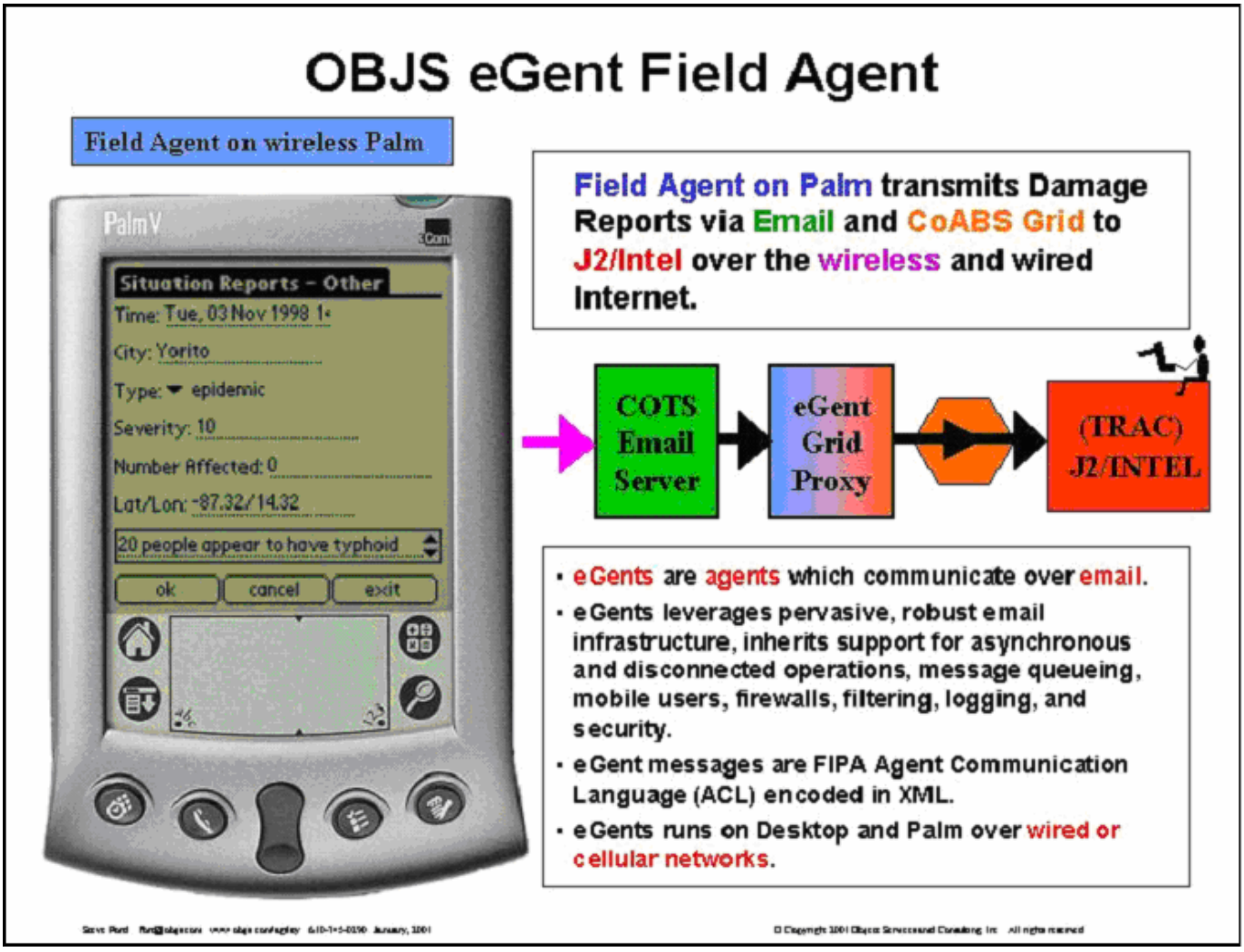
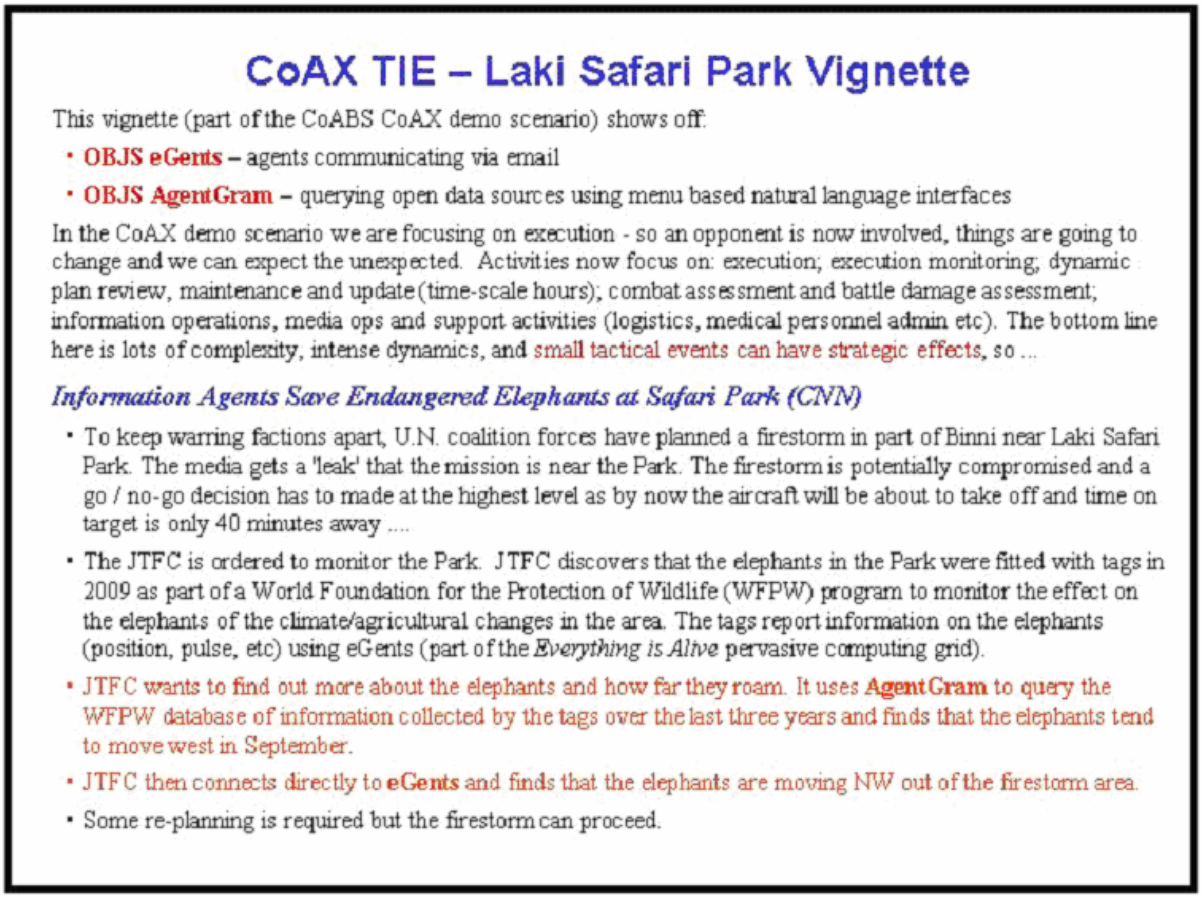


Figure 8: eGent field agent in the MIATA TIE

2.3.2 CoAX TIE

In the coalition domain (**CoAX TIE** - *http://www.aiai.ed.ac.uk/project/coax/demo/2001/*), eGents played a role in a biosurveillance vignette of a larger demo. It is the year 2012. Elephant herds in mythical Laki Safari Park in Binni, Africa, are migrating through a planned UN firestorm area. Both the UN and the press have just become aware of this, and care must be taken to locate the elephants and avoid harming them. A biosurveillance program at Safari Park had commenced in 1009, and eGents is being used to monitor the movements of large mammals including the elephants. The information is collected monthly and stored in a DBMS which can be accessed across the web (using OBJS MBNLI). An MBNLI query discovers the elephant migration path, and eGents is pulsed for the current location of the elephants, which is displayed on the CoAX MBP map. A last minute decision must be made about the placement of the firestorm to avoid harming the elephants. The scenario shows off bio-surveillance, eGents publish-subscribe, and finding and accessing a new data source with natural language commands. The CoAX eGents release used the free Apache James email server to permit demoing eGents in standalone CD based demos. We also added a simulation data source (to simulate the Elephant Status Monitor receiving data). We iterated with Austin Tate (U Edinburgh, overall coordination), Patrick Beautement (Qinetiq, demo scenario), David Allsopp (DERA, integration), and Craig Knoblock (ISI, an Ariadne information source wrapper), and others. This work was demoed standalone at the CoABS Workshop in Miami (January 2001) and integrated with other CoAX components at the CoABS Workshop in Nashua (July 2001). See **CoAX TIE avi** [*http://www.aiai.ed.ac.uk/project/coax/demo/2001/*].



Figures 9: CoAX TIE Vignette

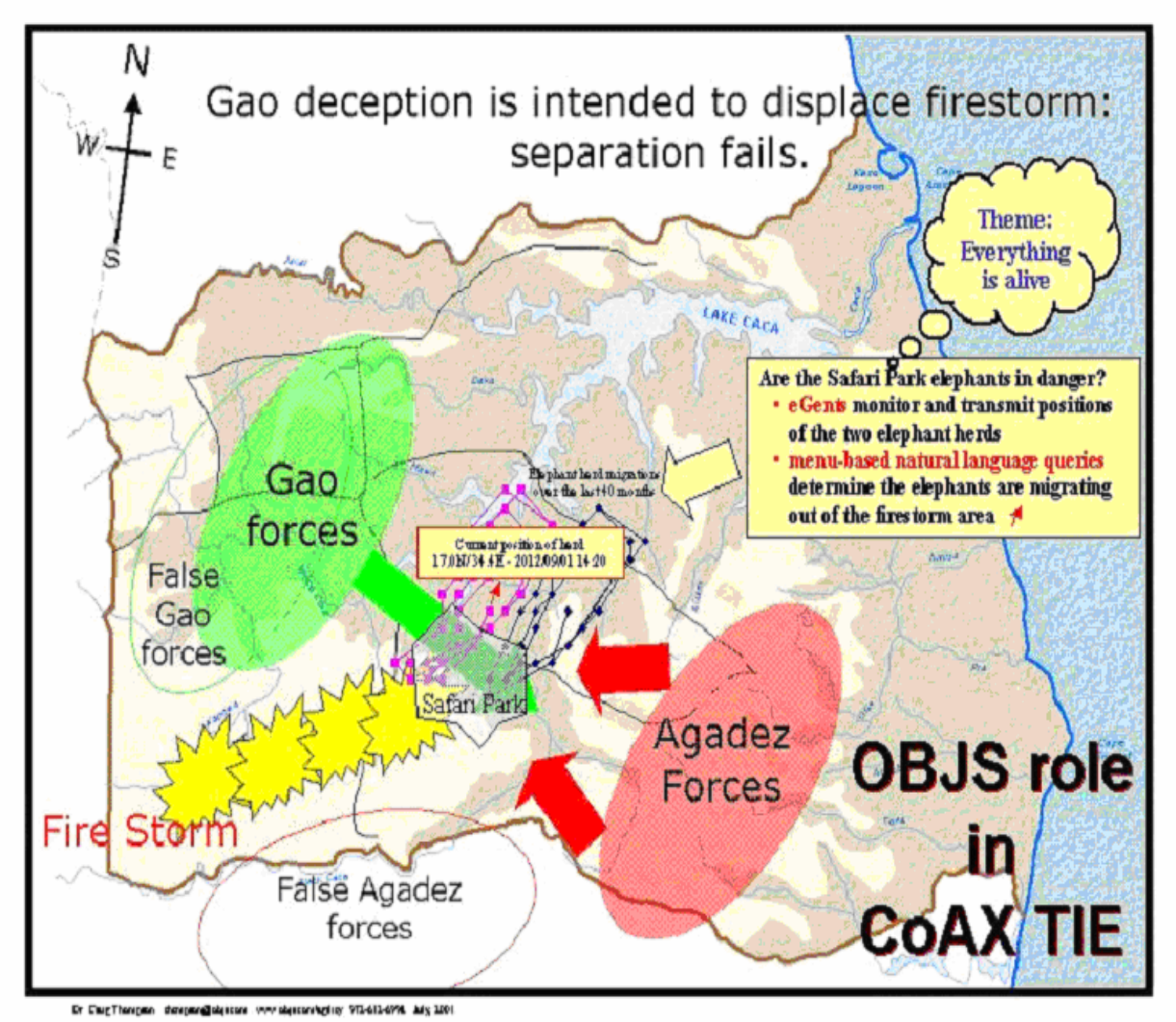


Figure 10: Map showing elephant migration as reported by eGents and MBNLI

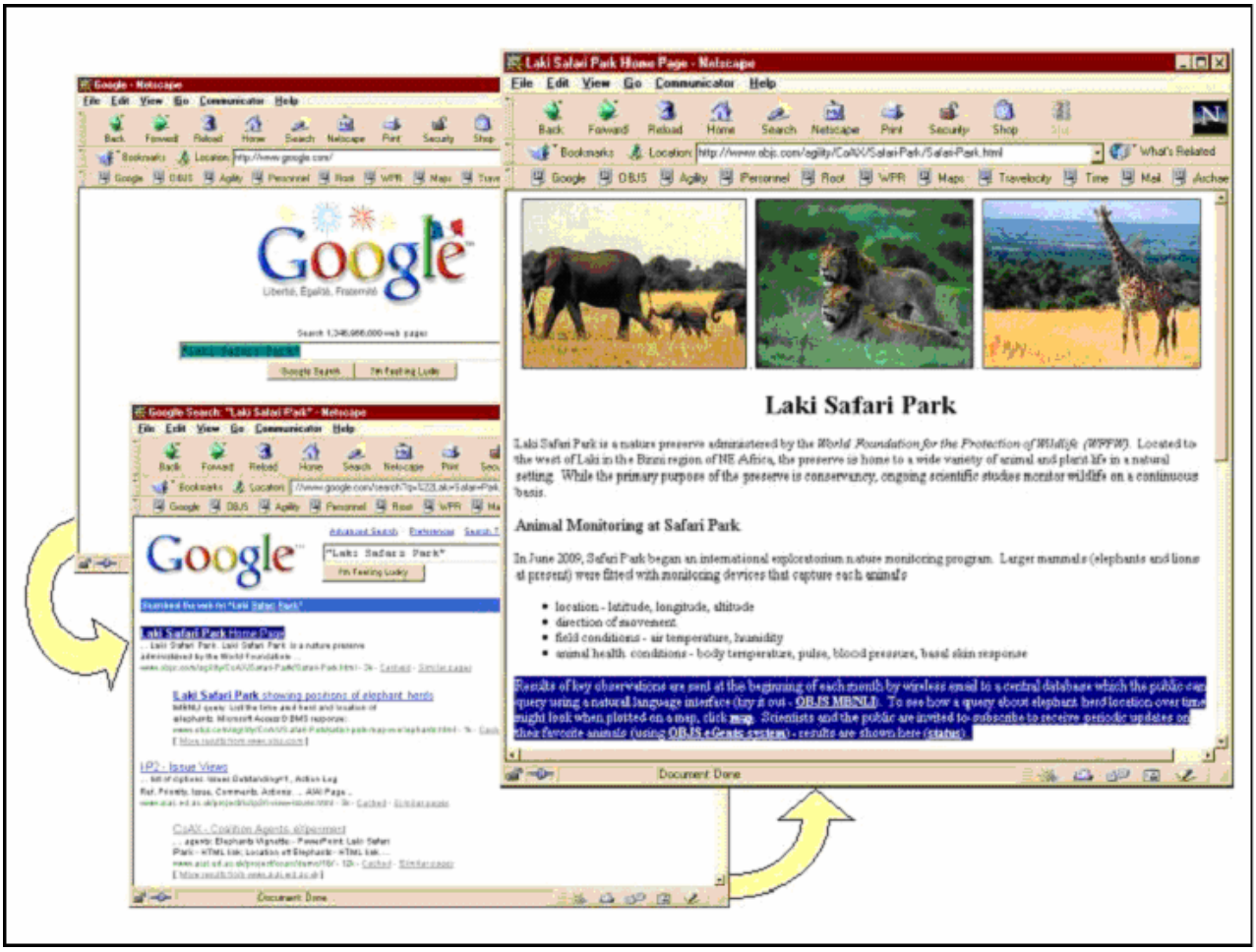


Figure 11: Finding MBNLI and eGents via a web page

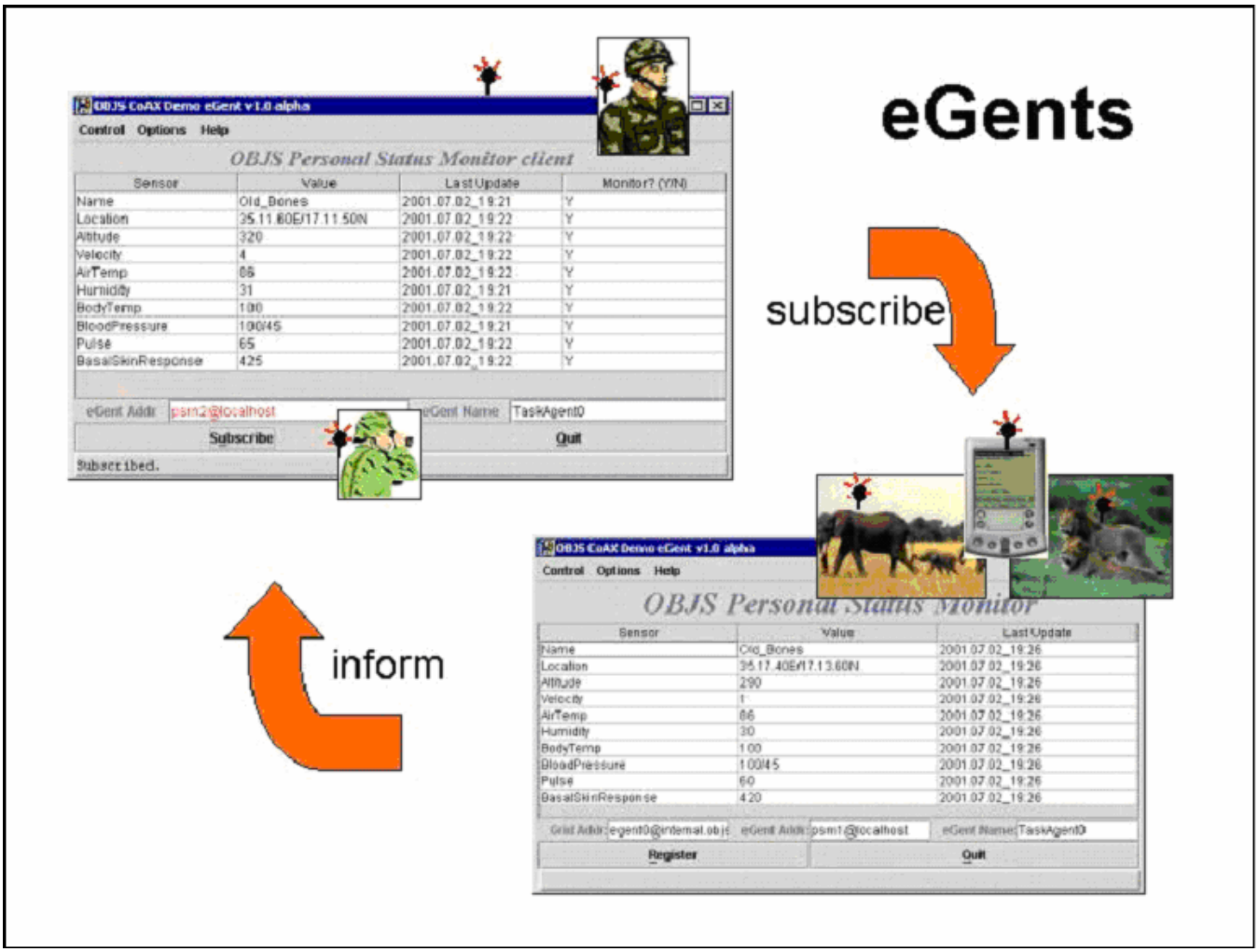


Figure 12: eGents reporting the positions of the elephants in Laki Safari Park

2.3.3 JBI TIE

In the small unit operations domain (JBI TIE), commanders drilled down via a map to subscribe to and monitor troop and platoon status during an attack, and fuselets noticed patterns (e.g., chemical attack, soldier wounded). This TIE was initiated in January 2001 at the suggestion of Jim Hendler (DARPA CoABS PM) and Dan Daskiewich (AFRL, in charge of CoABS), who were interested in understanding how eGents might interoperate with the Y-JBI project at Rome, being developed by Maj. Robert Marmelstein as part of the Joint Battlespace Infosphere project. Steps were:

- develop a demo scenario for Small Unit Operations - The demo begins when the Y-JBI system sends the commander an email describing a mole report of rebels shadowing US platoons in Bosnia. The commander zooms in via a map, and four platoons appear. The commander subscribes to them (Y-JBI communicates the subscription to eGents at the remote platoons). eGents periodically send Y-JBI platoon level personal status monitor (PSM) updates on individual troop health status. During the course of the demo, two platoons are attacked, one with conventional weapons and the other via a chemical attack. Y-JBI fuselets interpret the raw PSM reports to make this determination of an attack in progress and report to the commander the ongoing status.
- agree on a common ACL messaging format - This required some additions to eGents: support for additional ACL including time-constrained and periodic *subscription*s as well as additional *suspend*, *resume*, and *cancel* performatives.
- extend eGents with a simple simulation capability
- test eGents and Y-JBI communication between Plano TX and Rome NY, make an .avi movie, and demonstrate the result at the the CoABS Workshop in Nashua (July 2001). See JBI TIE avi (.exe includes TechSmith TSCC Codec and viewer - 2.9MB - 2:14 min - *http://www.objs.com/agility/tech-reports/0107-CoABS-Agility-Nashua/OBJS-CoAX-TIE-1024-768-256-TSCC.avi.exe*).

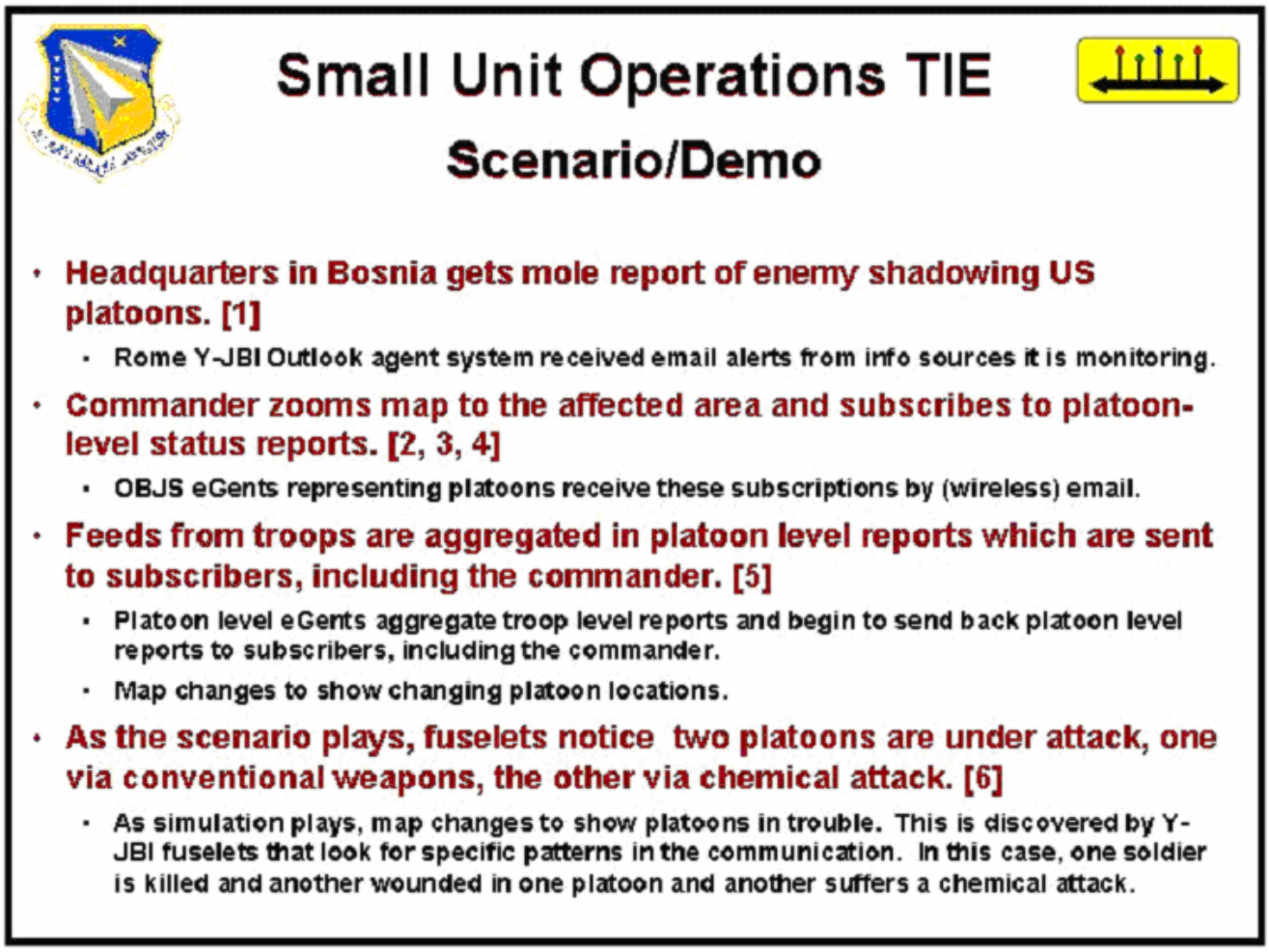


Figure 13: JBI Small Unit Operations TIE

Figure 14: Illustrating steps in Figure 13

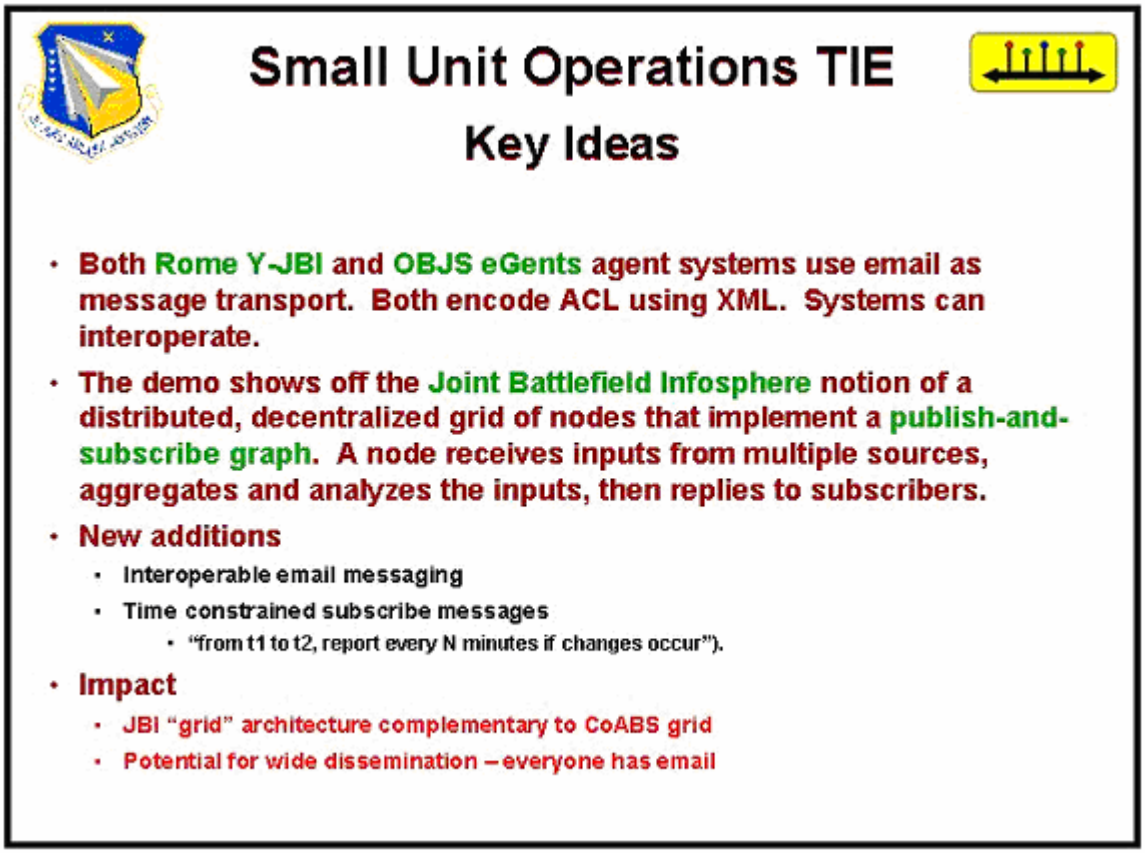


Figure 15: Key ideas in Small Unit Operations TIE

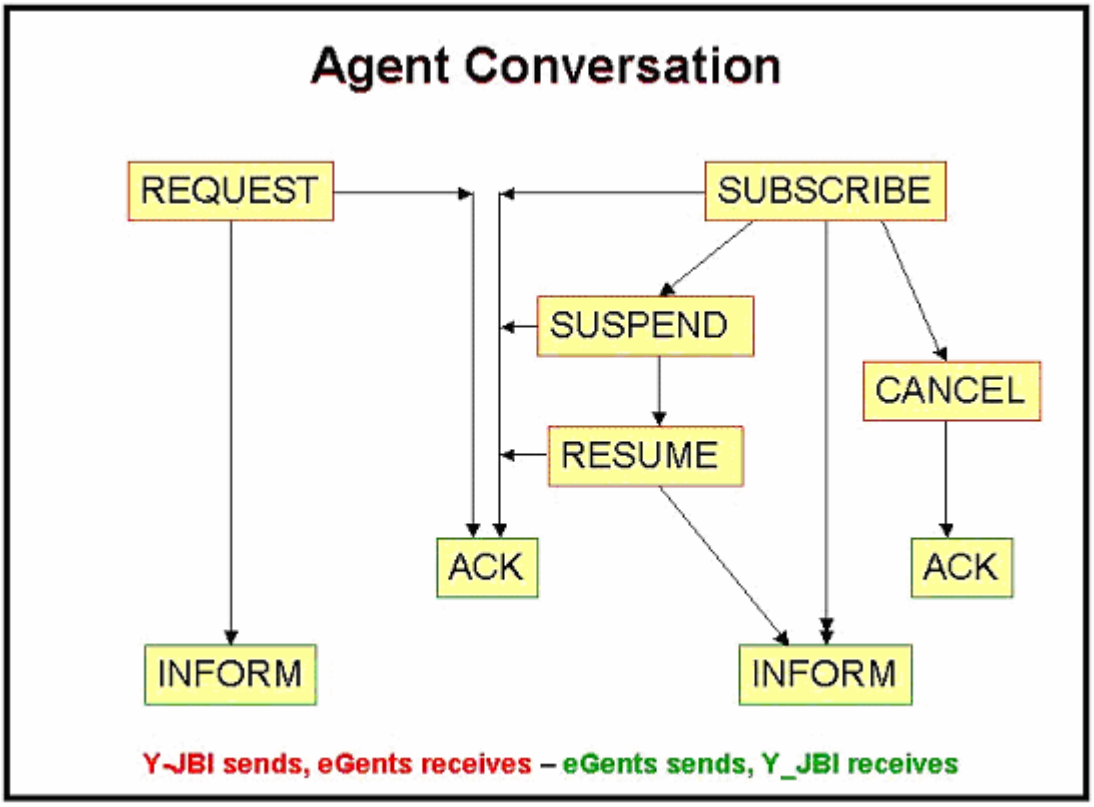


Figure 16: Agent Conversational Interactions

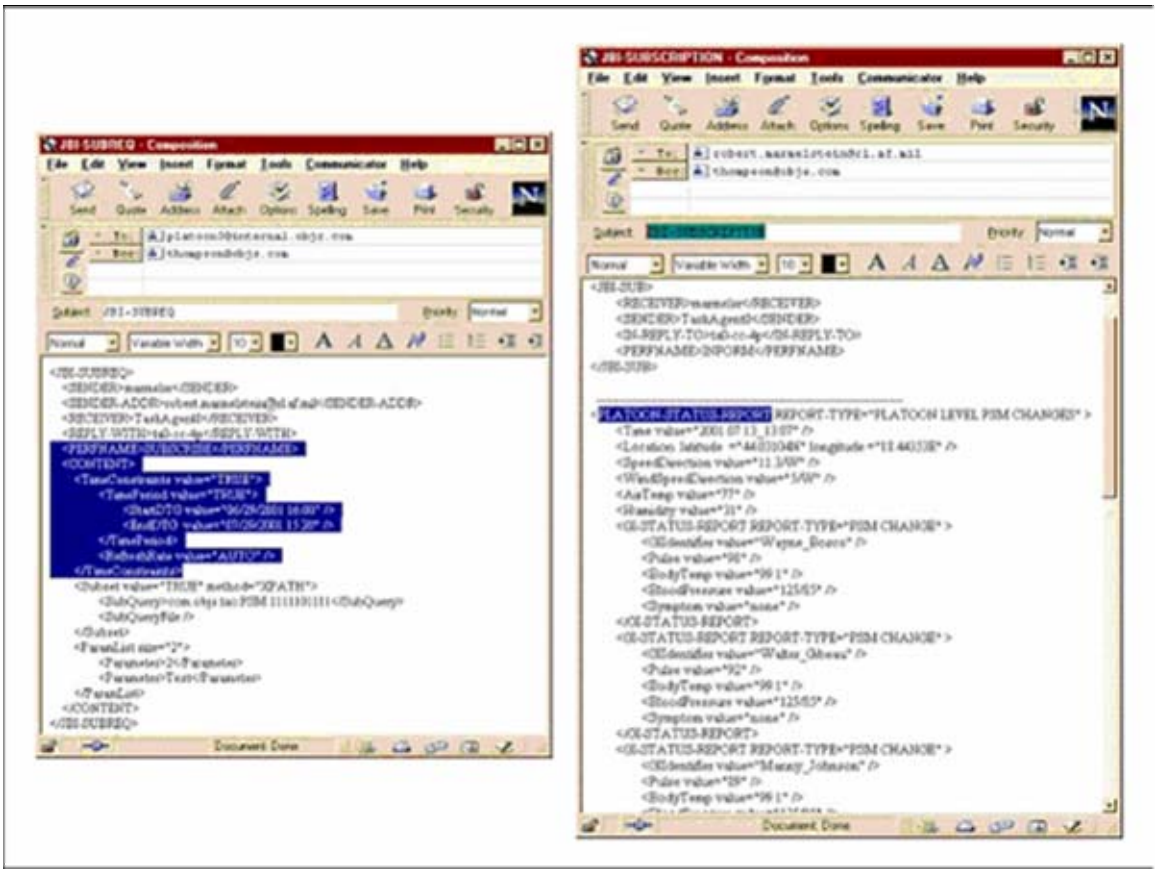


Figure 17: SUBSCRIBE and INFORM

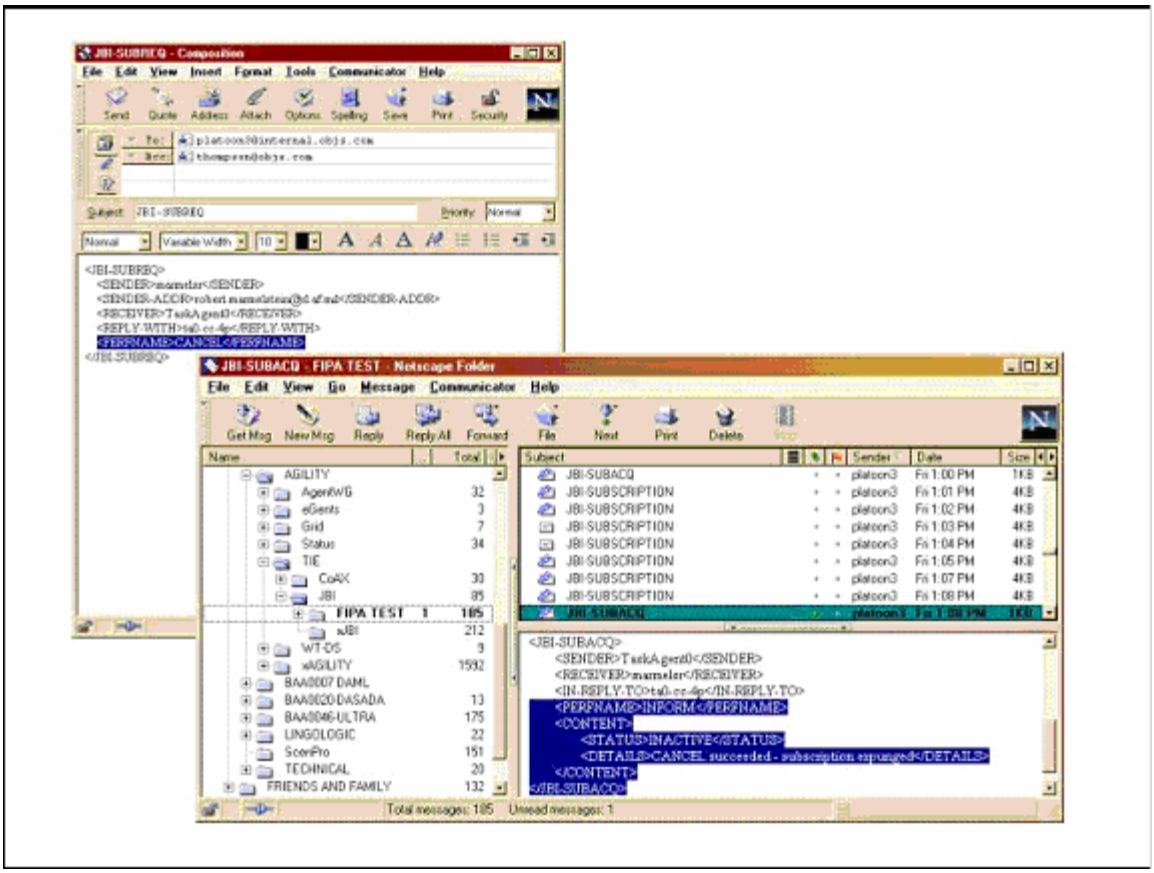


Figure 18: CANCEL and CONFIRM

2.3.4 Other Interactions

In September 2000, Gail Kaiser (Columbia) requested a copy of the Palm version of eGents for possible use in DARPA Dasada. In April 2001, we released eGents source to Joe Giampapa (CMU). He wants to get Retsina to interoperate with the agents that other CoABS research groups have made available "on the Grid" and wanted other research groups to be able to use his agents "via the Grid". The idea is that he'll get Retsina to access the PSM on the 7x24 Grid in the same way the PSM Grid Client does.
In addition, work has begun on eGents under CoABS has led to two additional contracts:

- the DARPA UltraLog [*http://www.darpa.mil/tto/*] MsgLog contract (February 2001) - the idea is to provide the ALP-Cougaar agent-based system with multiple communication transports (email and NNTP in addition to RMI) and then use policy management to select among them to insure survivable, robust message delivery in chaotic environments. David Wells described our Msg*Log project, including its origins in the CoABS Agility project, in Msg*Log: Email-based Agent Messaging to Improve Robustness in a Distributed Logistics Planner [*http://www.objs.com/agility/tech-reports/20010501-STC.doc*], a paper for the Software Technology Conference (STC 2001), Salt Lake City, Apr 29-May 4, 2001.
- an SBIR II with ScenPro as prime (October 2001) - the idea is to extend eGents in several ways to support small unit operations

2.4 Related Work

Agent projects that are related to eGents are JATLite, SodaBot, BeeGent and VisitorBot.

- JATLite implements a subset of eGents functionality (the ACL bus) using applet communication protocols as the transport. As applets are limited to communicating only with the host they were downloaded from, much of the JATLite infrastructure is geared towards building a peer-to-peer messaging bus out of this star communication topology (i.e., all applets talk back to a single node from which they are downloaded). This infrastructure includes components that perform performative queueing and routing function. eGents avoids this limitation by using e-mail, which is peer-to-peer by definition. eGents reuses the email store-and-forward capability (that mailboxes inherently provide) to implement performative queueing and routing. The centralized nature of JATLite raises some scalability concerns in serving large communities of agents. eGents has no single locus of control, and relies on the scalability of email. The JATLite feature that agents have to be applets can be restrictive and unnecessary. eGents can support batch agent applications that do not require the interactivity that applets provide, and conversely need greater access to the host platform than applets provide. The security restrictions of applets makes them hard to operate over corporate firewalls. eGents e-mail based ACL bus imposes fewer requirements on agents that use it, and firewalls do not pose a problem for an e-mail based ACL bus. While applets limit

JATLites range of applications, they also make JATlite easy to deploy and distribute. JATLite uses web browsers as the agent platform and applet downloads as the software distribution mechanism. We would like for eGents software distribution to be as easy as downloading an applet, without the downside's of restricting agents to be applets.

- SodaBot [Sodabot Home Page - *http://alpha-bits.ai.mit.edu/people/sodabot/*] provides a toolkit for building personal online assistants (electronic secretaries) and application agents (coordinate tasks or information transfer amongst people). It uses an e-mail based agent communication infrastructure, with a Perl-based ACL (and content language). SodaBot has a built-in software distribution mechanism to disperse code blocks from a static agent specification to the locations they need to execute at. SodaBot has similarities to eGents, but does not use widely supported implement platforms such as Java and XML.
- AT&T's VisitorBot is more an application than a platform, but relevant in that it also uses e-mail as a computational infrastructure to automate meeting scheduling amongst multiple lab members. Their application uses a limited communication language between desk top agents (one which is transported over e-mail), and a star architecture where all desktop agents communication to a central meeting scheduler agent. VisitorBot does have the intelligence to find the user (if not at his workstation, page him, or fax him.....), something which is worth incorporating into eGents.
- BeeGent from Toshiba Labs [*http://www2.toshiba.co.jp/beegent/index.htm*] seems similar in goals to eGents. Like eGents, BeeGent has an ACL-over-XML bus, but over HTTP instead of email. They apparently have a mobility solution over http as well. It is unclear why (or whether) they do not use Koala and XML-based mobility. They also focus on agents that are java components, and have a component wrapper language. Their focus seems to be more on enterprise-wide access to repositories and less on office automation. This might influence our respective architectures differently.
- Zaplet email is used to move links to javascripts between users [appmail zaplets - *http://java.sun.com/features/2001/08/appmail.p.html*]. All the real computing is going on on the web server. You go to their site and choose from a library of zaplets, say a poll zaplet, you fill out the zaplet form (a form letter), and their server sends it to the recipients you specify. The recipients get a message containing a link to the client version of the poll on their web server. They fill out their part and their web server updates the server version of the poll, on the web server. Optionally, it might send you a message saying it has received a response. You go back to their server to check the results. It *looks* like the poll is being updated in your local mail archive if you keep the original message that's sent out, but its not really, its just that the message doesn't really have the script or data in it, it has a link to it on the Zaplet server. In contrast, eGents uses a thin agent platform on the client side to connect (Java) applications on the local machine to communicate (using email as the transport) to similar applications using similar eGent clients on other machines. There is no central server. So, the mechanism is totally different, but both mechanisms could implement the applications they are targetting. A disadvantage to their design is that it is dependent on the Web, and connectivity to their server, so would not work with the same level of disconnectedness that eGents can. The advantage is that its fairly independent of any software on the

client, especially all the security issues involved with running Java applications on clients. Both technologies can be used to create back office or interactive apps connected into email. The eGents approach is more decentralized, usually means more scalable, and also works disconnected.

2.5 Future Directions

The following make sense as next steps for the eGents project - some are research tasks and some are engineering aimed at making the system more usable.

- handle dangling subscriptions (subscriptions from dead clients)
- integrate eGents into the Grid as an alternate transport
- integrate Java security and make eGents' security policy easier for the user to understand and trust
- add XML-based mobility solution (e.g Koala from Inria)
- make eGents as easy to install on the desktop as a plug-in or applet (without the limitations of either).
- tackle some tedious but tractable engineering challenges allowing eGents to share a user's mailbox without him/her noticing the email pollution.
- eGents + WebTrader + ... = e-mail-based Jini
- explore adaptive application architectures that make use of agent mobility. Right now, mobility is used at the individual agent level, without any collective intelligence driving it.
- XML'ize the AgentMap file format. This will save us from building custom parsers, if the microscripting language becomes more complicated.
- improve the performance of the Palm version of eGents, and merge in changes from the Java version.
- prepare a general public release of the Java and Palm versions of eGents

2.6 Summary

Problem. Most agent systems rely on unique and extensive infrastructures that must be widely available if the systems are to be widely used. This is an obstacle to the widespread adoption of agent technology.

Objective. eGents demonstrates that agent technology can scale to global proportions by leveraging an existing infrastructure, rather than propagating a new one.

Approach. eGents is, basically, agents communicating over email. Email is pervasive and robust and already provides many functions an agent system needs - message queuing, encryption, filtering, firewall traversal, support for mobile users. The eGent platform is a multi-threaded Java app with access to an email account on an SMTP/POP3 server. Individual eGents are identified by their name and the email address of the platform, and are messaged as threads in the eGent platform's process. Messages are in a subset of FIPA ACL (subscribe & inform), encoded in XML, and sent as the body of an email message.

Recent Progress. **(a)** We ported eGents to a wireless Palm using Sun's newly released Java for devices, KVM, a troublesome port because of KVM's bleeding-edge nature, lack of tools, and because most Java code will not run under KVM, including most of Sun's Java class libraries, and all of the third party code on which eGents depended. The result runs, but is slow. The main benefit is that it should port easily to other devices (set top boxes, smart phones). **(b)** We built the eGentGridAgent proxy which extends the grid by demonstrating interoperability of GridAgents with agents that are less "connected". We could have just made a proxy for the PSM demo, but spent a little more time coming up with a generic proxy. We didn't want the eGents platform or an individual eGent to know anything about the Grid, or vice versa. The only component of the system that knows anything about both is the eGentGridAgent, which is just an eGent to eGents and just a GridAgent to the Grid. We wanted to keep the protocol situation simple. We didn't want an eGent to have to know LAN protocols, like Jini or HTTP or RMI or anything that made the eGent less mobile, disconnected, and asynchronous. So, an eGent can be "on the Grid" without knowing anything about the Grid. **(c)** We installed the Java version of eGents on the 7x24 Grid including a documentation/installation page. **(d)** We explored another alternative for integrating eGents and the grid as a LAN-to-LAN bridge by extending the grid to provide multiple native communication protocols, e.g., not just RMI but also eGents. **(e)** We extended eGents' subscribe performative with temporal constraints and added *Suspend*, *Resume* and *Cancel* performatives. **(f)** We inserted eGents into the MDATA, CoAX and JEI TIEs.

Demonstrations. **(a)** The initial eGents demonstration was a technical report subscription service. **(b)** For the Science Fair, we developed an eGents demo for a NEO-like scenario. We assume that NEO ensures have Personal Status Monitors (PSMs) that monitor location, medical status, threats, etc. These PSMs contain eGent platforms and periodically exchange status updates with subscribers, e.g., command ports, medevac units, state department monitors. **(c)** For the MDATA disaster recovery TIE, we implemented Field eGents sending Damage Reports to JOINTEL via email and the Grid. **(d)** For the CoAX coalition forces TIE, we drove a surveillance demo where elephants in Safari Park in the Ekorafire zone are in possible danger which requires replanning CoAX operations. **(e)** For the JEI small unit operations TIE, we connected eGents to the Rome V-JEI Outlook email agents project in a scenario that shows a commander receiving a mole report about enemies shadowing US troops, then drills down, subscribes to the platoons and watches the action unfold. JEI fuselets identify platoons in trouble, one from conventional attack, the other from chemical attack.

Technology Transition. **(a)** We participated in the TIEs mentioned above. **(b)** The encoding of FIPA ACL in XML and the JavaMailMAPI API are candidates for standards, so we submitted these to the FIPA 99 Call for Proposals.

System Requirements. **(a)** eGents should run on any Java platform, but has only been tested on Windows NT 4.0 SP3. Requires Sun's Java JDK 1.2.2, 5MB of free disk space, and an email (SMTP/POP3) account for each machine on which the eGents platform will run. **(b)** eGents also runs on KVM (J2ME CLDC 1.0 FCS) on the Palm Vx, the PalmOS Emulator 3.0a5, or under Windows NT. Wireless communications from the Palm is currently via the Novatel Minstrel V wireless modem, Omnisky's wireless Internet access, and AT&T's CDPD digital cellular network. **(c)** The eGentGridAgent has been tested with CoABS Grid v1.5.0 beta.

Potential Directions. **(a)** update to latest gridrelease and prepare a public release of the Java version, **(b)** speed up eGents on the Palm under KVM if possible, else port to C++, **(c)** prepare a public release of Palm version, **(d)** demonstrate email-based eGent installation, **(e)** integrate Java security and make eGents' security policy easier for the user to understand and trust, **(f)** demonstrate eGents that cross firewall boundaries.

Figure 19: Summary

Chapter 3

AgentGram Prototype: Natural Language Interface for Agents

3.1 Objective

The objective of the AgentGram project was to develop a modular *menu-based natural language interface* (MBNLI) component that can be used in client-server web environments as a front-end to agents and other Internet resources (e.g., data sources). From the point of view of Agility's goal to demonstrate agent capabilities that scale to mass markets, the impact we were aiming at was to develop a technology that enables humans, anywhere on the semantic web, to task and query remote agents and Internet resources using complex but understandable commands in constrained natural language.

Figure 20: AgentGram Overview

Specific scientific and engineering subgoals were:

- enable MBNLI on any web page as a way to communicate with remote web resources, e.g., agents, databases, ...
- semi-automate generation of MBNLI interfaces
- prototype a companion speech interface to MBNLI
- develop MBNLI as a component that can interface to other components and can connect to the CoABS grid
- demonstrate MBNLI in scenarios of interest to DoD

Our primary thesis is that MBNLI can operate on the desktop or web to provide naive users with natural language interfaces they can actually use. Thus, MBNLI is a significant potential step towards the development of a more semantic web and to "scaling agent systems to the masses."

3.2 Background

Menu-based natural language interface technology (MBNLI) combines constrained grammars, a predictive parser, and interface technology to provide users with a guided natural language query and command capability. As explained in **MBNLI Overview** [*http://www.objs.com/agility/final/prototype-AgentGram/docs/0101-MBNLI.doc*], this approach bypasses frustrating habitability problems that other NL interface technologies suffer from where users undershoot or overshoot the NL systems' capabilities.

3.3 Technical Accomplishments

Under the DARPA CoABS contract, we extended MBNLI in the following ways:

3.3.1 Initial AgentGram Prototype

An initial AgentGram prototype, developed in 1999, was based on the notion of distributed agents which contain grammars that can be dynamically composed. Users were able to construct (using cascading menus) complex sentences (commands/queries) which simultaneously involve the grammars of several agents. The grammars were dynamically loaded from web-based agents on demand. This first implementation focused on dynamically constructing restricted English phrases from the partial grammars of multiple distributed agents simultaneously. The result is a readable sentence which represents a complex executable command. See screenshots of this **AgentGram** prototype [*http://www.objs.com/agility/final/prototype-AgentGram/docs/ 9904-agentgram-screenshots.html*]. This implementation was somewhat simplistic, using tree grammars represented in XML. Later implementations extended the MBNLI toolkit which permits attributed context free grammars.

3.3.2 Web-ready MBNLI

This task was the heart of the AgentGram project. The objective was to enable humans anywhere on the semantic web to task and query remote agents and Internet resources using complex but understandable commands in constrained natural language. The interfaces appear as annotations on web pages. The system should scale to any number of users, grammars, web pages, and target resources. The system should be deployable with no effort by the user (no explicit downloading action). This is a step in making agent technology pervasive. Making MBNLI web-ready, required re-engineering several parts of the original system:

- We re-implemented the front-end user interface. We added the ability to support alternative interaction paradigms including cascaded menus (re-implementing the Java Swing menus placement algorithm for cascading menus) and phrase buttons (an alternative interaction paradigm to minimize screen real estate).
- We worked on having thin MBNLI interfaces (little download and no install overhead so no barrier of use). We considered several approaches - refreshing whole pages, applets, and downloading the entire parser. The first approach appears awkward. The last is OK for demos and for users that want full service but not for establishing wide-spread adoption by end-users.
    - We built a prototype applet that handles menu selection but where the parser is remote. Initially we used two-way RMI but found that that involved applets signing certificates and that it violated browser security. We looked into executing the RMI version of the applet in IE (which required downloading and installing IE5.0) but the applet wouldn't run and IE's console window provided a cryptic message. We redesigned the applet to eliminate two-way RMI. We downloaded and installed TinyWebServer, a 48k HTTP web-server, so he could test the MBNLI Applet. We completed an initial working applet with expert support. Different portable specs are accessed via different applets which are parameterized with information about the portable spec.
    - We later implemented a stateless C-based (cgi) front-end to the parser which generates HTML/Javascript, no Java at all. It provides an interface much like the applet version and has support for experts, but it is smaller and faster.
- We designed and implemented a web-based multi-threaded MBNLI parser farm enabling parsers to be instantiated on the fly. The parser farm manages the set of active parsers and routes user requests to the appropriate parser based on the grammar and lexicon requested. If such a parser doesn't exist, then a new parser is started. A basic security model was also implemented.
- We developed a grammar-on-the-fly capability for MBNLI. This allows the user to select and change between grammars after MBNLI is up and running, rather than requiring this information at start-up. We added APIs to the parser and modified the NLI UI by adding menus and file dialogs to permit selection of portable specs.
- We developed a browser-based XML-driven dynamic interface for MBNLI. Internet Explorer allows page updates without refresh by supporting data-driven components.

This approach enables MBNLI to use a single static interface within a browser as opposed to refreshing for every change (involving the server).

- We re-implemented the experts API, adding new associated classes, and creating several new experts (code and UIs to facilitate and constrain user input) which can be invoked via definitions in the portable spec.
- We extended the web version of MBNLI to support remote query execution and local display of results using PHP [*http://www.php.net/*]to handle CGI <-> ODBC.
- We modified MBNLI to work on Win95 and NT machines with Winsock2 installed. This involved converting the parser/lincoo from Unix/C++ to Win95/NT MSVC, eliminating dependence on cygwin DLLs, and developing code to support Win32 sockets to allow them to be treated exactly like files. This reduced the backend codebase from 900k to about 338k.
- We benchmarked MBNLI and made various other improvements. However, more work is still needed here before we get a good overall picture of how to scale the design to 100s or 1000s of agents simultaneously using the parser across the web.

3.3.3 MBNLI Interface Generator for DBMSs

A sub-problem in making MBNLI widely useful is generating new MBNLI interfaces. If this requires specialized knowledge, it will slow down the process of scaling the technology for widespread use. The initial AgentGram prototype described above provides a simple way to do this for very simple tree structured grammars represented in XML. This is simple enough for many web developers to use as is. The original MBNLI prototype provided a grammar parameterized with DBMS elements stored in a .spc file in Lisp syntax but creating such files was tedious, error prone, and required a Lisp background.

```
(defrel Elephant
    :key-attrs (Name Time )
    :default-attrs (Name Location Altitude Velocity AirTemp Humidity BodyTemp
                    BloodPressure Pulse BasalSkinResponse Time Herd )
    :menu-string ((:default "elephants")
                  (of "of elephants")))
(defattr Elephant Name
    :type STRING
    :menu-string ((:default "elephant's name")
                  (:short "name")
                  (:plural "names")
                  (:whose-is-default "where the elephant's name is")
                  (:whose-is-short "whose name is"))
    :op-prop ( :comparable :groupable)
    :trx " Elephant.Name"
    :expert "DBCHOICE RelName=Elephant AttrName=Name")
...
```

Figure 21: A Fragment of a Portable Spec in Lisp from the CoAX TIE

A first step was to develop an equivalent XML representation, as shown in the following example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE grammar PUBLIC "-//OBJS//DTD NLI-PSEditor//EN" "file://C:/MBNLI/PSE/portableSpec.dtd">
```

```
<portableSpec >
  <relation  name="Elephant">
    <attrInfo >
      <keyAttrs >
        <attrName  name="Name"/>
        <attrName  name="Time"/>
      </keyAttrs>
      <defaultAttrs >
        <attrName  name="Name"/> <attrName  name="Location"/> <attrName  name="Altitude"/> <attrName
name="Velocity"/> <attrName  name="AirTemp"/>
        <attrName  name="Humidity"/> <attrName  name="BodyTemp"/> <attrName  name="BloodPressure"/> <attrName
name="Pulse"/>
        <attrName  name="BasalSkinResponse"/> <attrName  name="Time"/> <attrName  name="Herd"/>
      </defaultAttrs>
    </attrInfo>
    <menuStrings >
      <menuEntry  type="default"  name="elephants"/>
      <menuEntry  type="of"  name="of elephants"/>
    </menuStrings>
    <relAttrs >
      <relAttrChild  name="Name"  type="STRING"  txString=" Elephant.Name"  expert="DBCHOICE RelName=Elephant
AttrName=Name">
        <menuStrings >
          <menuEntry  type="default"  name="elephant's name"/>
          <menuEntry  type="short"  name="name"/>
          <menuEntry  type="plural"  name="names"/>
          <menuEntry  type="whose-is-default"  name="where the elephant's name is"/>
          <menuEntry  type="whose-is-short"  name="whose name is"/>
        </menuStrings>
        <operator  name="comparable"/>
        <operator  name="groupable"/>
      </relAttrChild>
...
```

Figure 22: A Corresponding Fragment of a Portable Spec represented in XML

Then, to largely automate the process of quickly developing MBNLI interfaces to DBMSs, we developed the PSEditor GUI. The PSEditor enables MBNLI users to quickly create and edit specifications used by MBNLI to generate MBNLI interfaces to tables in a relational DBMS. The PSEditor uses the XML-based format for SQL-related portable specifications. It also accepts the original Lisp syntax. PSEditor is composed of about 60 Java classes and is about 131K (source code size). PSEditor GUI can be useful for desktop or Internet-based deployment of MBNLI. A screenshot of the PSEditor editing the NEO TIE tables is shown below.

The final step was to develop a utility to read RDBMS catalogs. Amazingly, searching many documents, KBs, and posts to newsgroups turned up nothing of use, and there does not appear to be a standard catalog export format or utility. After considerable experimentation, we completed an ODBC schema import and translation capability so that, in a portable way (so far tested with Oracle 8 and Microsoft Access), database schemas (tables, columns, primary keys, and joins) exported from a relational DBMS can be used to automatically define initial natural language interfaces for use with MBNLI. The capability has been integrated with the MBNLI Portable Specification Editor, which allows editing of the generated interface and translation to the MBNLI portable specification format (see **.avi movie** - *http://www.objs.com/agility/final/prototype-AgentGram/docs/0002-agentgram-ODBC-import-movie.avi*). This allows the rapid creation of new AgentGram interfaces by relatively naive users.

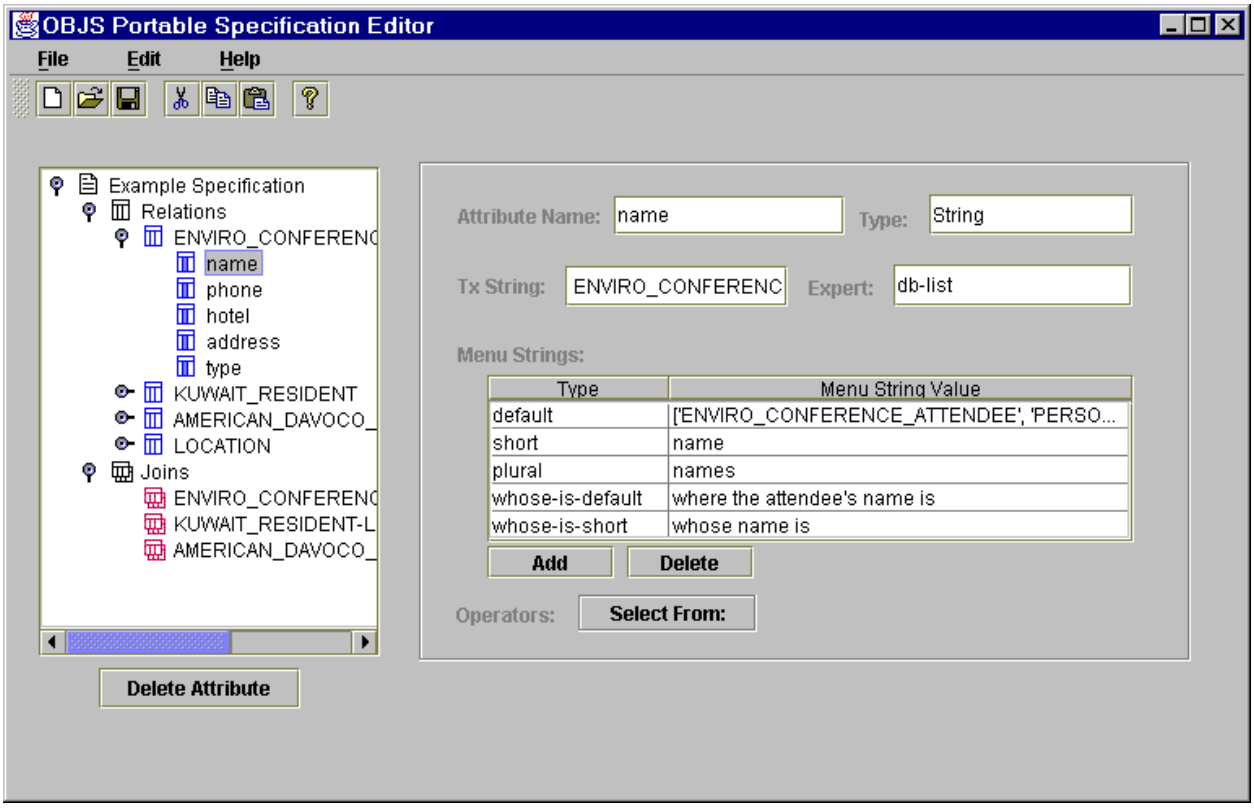


Figure 23: Portable Spec Editor

3.3.4 Speech Interface

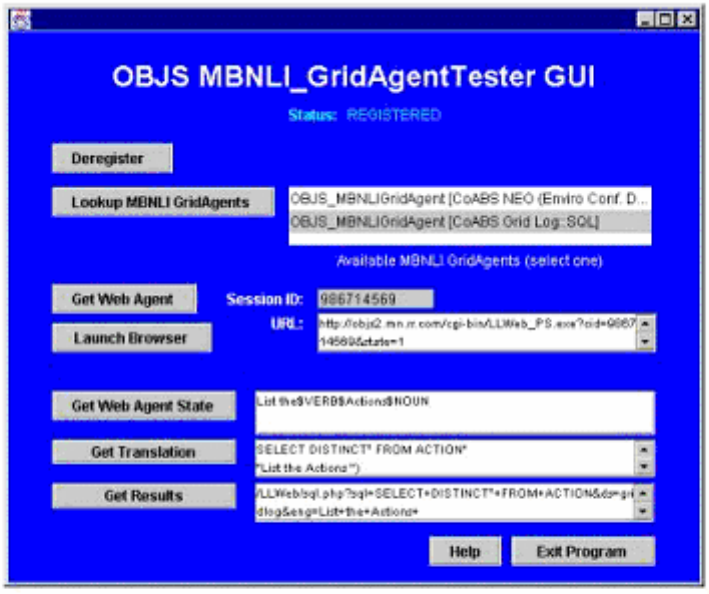
We also wanted to support a speech interface to MBNLI so speakers could simply read the menu choices. We reviewed **W3C Voice Browser standards** [*http://www.w3.org/UI/Voice/1999/voice-activity-brief*] and then considered several commercial speech interfaces, including JavaSoft's **Java Speech API** [*http://www.javasoft.com/products/java-media/speech/forDevelopers/jsapi-guide-0.7/index.html*], IBM's **ViaVoice Technology** [*http://www.software.ibm.com/viavoice*], and IBM's **Speech for Java** [*http://www.alphaworks.ibm.com/formula/speech*] (the only one of these products to support a Java API at the time). We fairly rapidly completed a rough proof-of-concept integration of MBNLI and IBM Via Voice based on IBM's Speech for Java API with grammar rules dynamically defined using Sun's proposed grammar standard JSGF. This enables users to compose sentences using speech or via menu selection.

3.3.5 Gridifying MBNLI

CoABS Grid. The CoABS grid is a JINI-based implementation of an agent interoperability platform developed by GITI, the DARPA CoABS program integration contractor. It is an important, on-going experiment in agent system interoperability. As described elsewhere, we

contributed architectural ideas to the grid. But in addition, we developed three standalone agent components (eGents, WebTrader, and AgentGram) that can play a role as grid components or services. As part of the Agility AgentGram project, we developed the the grid-relevant capabilities described below. *At the same time, we note that AgentGram can also standalone as a potentially pervasive capability that could be tied into any future grid implementation.*

- **MBNLI Interface to Grid Log.** For the CoABS Science Fair in November 1999, we demonstrated an interface to the CoABS grid log that allowed users to query the log files via MBNLI. This was done by first defining a database import facility to import the CoABS grid log files from XML into an Access relational DBMS, then developing an associated schema.
- **MBNLI Grid Agents.** We developed the following Grid agents, demonstrated at the CoABS Boston meeting:
    - **MBNLIGridAgentTester** - this agent has a GUI and can register or deregister itself on the grid. Once registered, it asks for all AgentGram agents on the grid and lets the user choose one, then establishes an AgentGram session
    - **MBNLIGridAgents** - these are remote agents, one per AgentGram interface (e.g., one for the DAVCO DBMS, another for the Grid Log interface). These agents offer a programmatic interface for controlling a session, accepting messages to get parse state, translation, and results of an execution.
    - **BrowserAgent** - pops up a Netscape or IE browser on the user's machine to permit the user to query the selected MBNLIGridAgent and see query results.
- **Launch Page.** In October, 2000, we created a launch page capability for the 7x24 grid - the page describes MBNLI and allows users to launch MBNLI demos. In November 2000, we converted MBNLI to function on the then latest version of the grid. MBNLI agents were maintained for a year on the **24x7 Grid** (see the grid archives available at that web site - *http://agent1.globalinfotek.com/status.html*).



1. Register/deregister MBNLIGridAgentTester on the grid.
2. Lookup available MBNLIGridAgents - two are found (interfaces to NEO DAVCO DBMS and to grid log).
3. Select one and get a session ID/URL.
4. Launch a (Netscape) browser to interact with remote MBNLI (LLWeb) parser farm. At this point, you can get the current state of the LLWeb interaction, whatever it is (state, translation of the state, or the url for the results [if you evaluate the url (in a browser) you will execute the query & the results will be returned.

Figure 24: MBNLI Grid Agent Tester

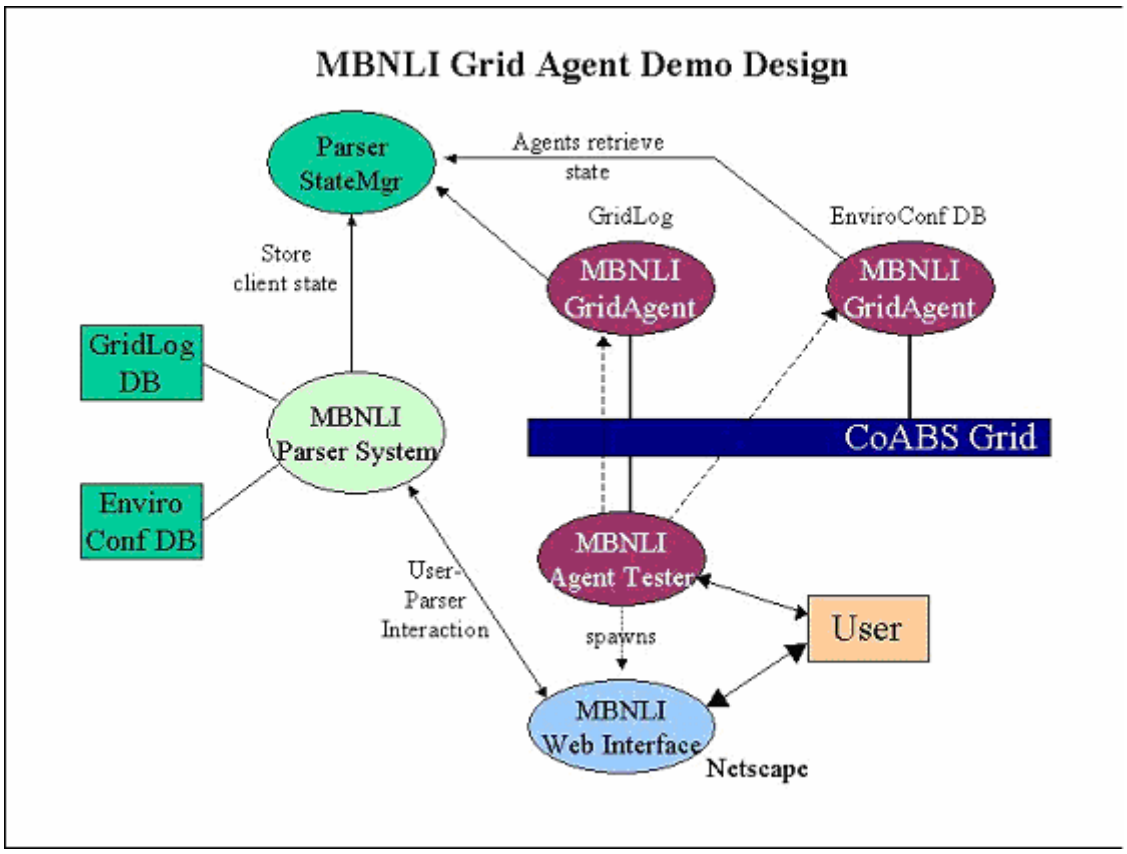


Figure 25: MBNLI Grid Agent Demo Design

3.4 Technology Transition

We demonstrated evolving versions of MBNLI at all CoABS PI Workshops and the CoABS Science Fair.

We applied MBNLI in the following DARPA CoABS Technology Integration Experiments (TIEs):

3.4.1 NEO TIE

The Non-combatant Evaluation Order (NEO) TIE involved an urban rescue effort and served to organize many CoABS program activities in the first year of the CoABS program. The thrust was on agent interoperability and rapid assembly of heterogeneous agent systems to solve problems. Lessons learned and components from this exercise where later incorporated into the CoABS grid.

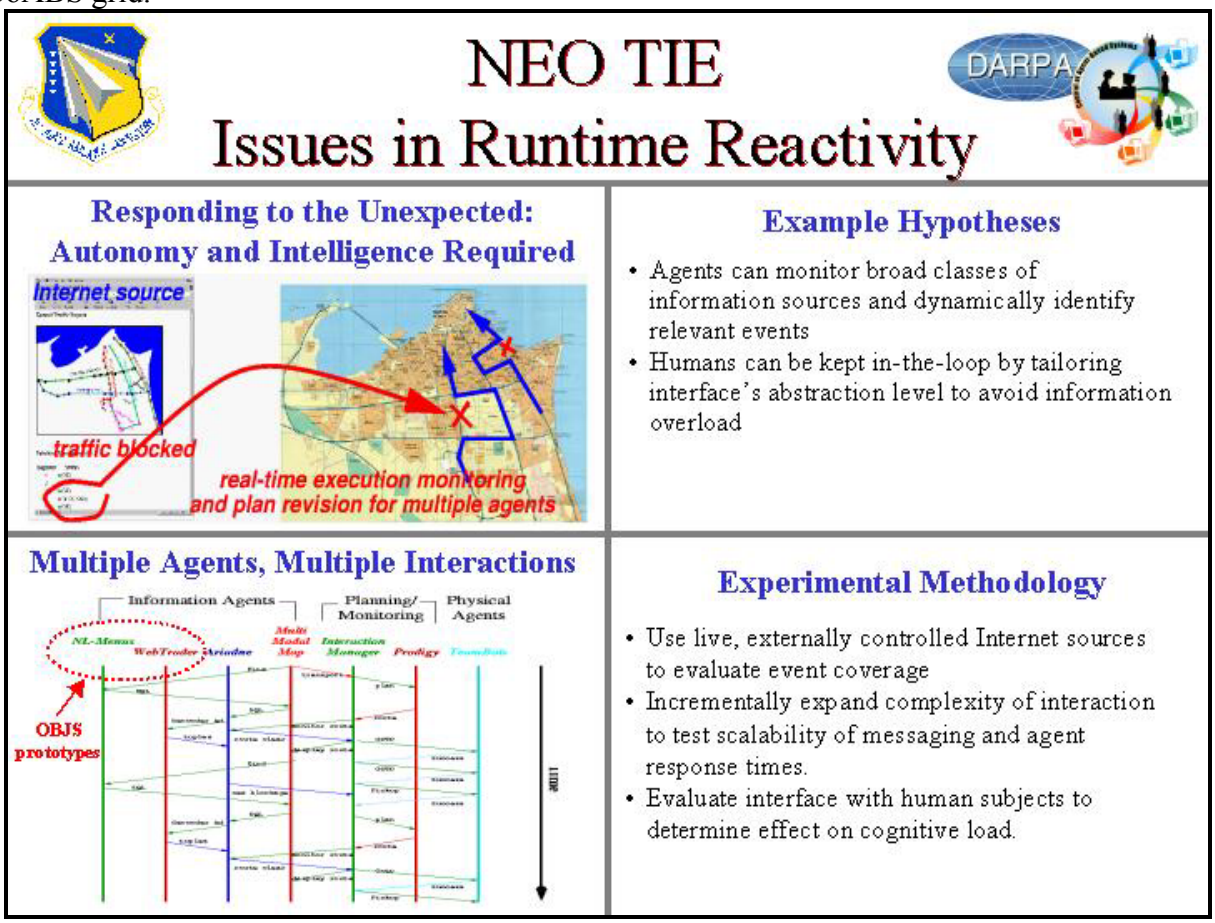


Figure 26: MBNLI in the NEO TIE

Paul Pazandak (OBJS) participated in a NEO TIE organizational meeting held at ISX in Agoura Hills (Los Angeles area) on 28-29 September 1998. Based on the meeting and subsequent discussions, we determined that MBNLI should play a role, and worked with Steve Minton (ISI Ariadne, TIE#2 coordinator) and Adam Cheyer (SRI Open Agent Architecture/Multimodal Map aka MMM) to develop a vignette (TIE #2) involving **Find Civilians, Get Them to Embassy** [*http://www.objs.com/agility/tech-reports/9809-TIE2-draft.html*]. In this vignette, OBJS MBNLI was used to query relationally formatted data. OAA/MMM was used to provide a speech interface and as a general controller. USC/ISI Ariadne was used to extract data on civilian locations from various web resources into a relational format. Minton supplied a relational schema that we used this to parameterize MBNLI to define a restricted language interface to the Ariadne data. The interactions between Ariadne, MMM, and MBNLI (as well as OBJS WebTrader) are shown in the figure below.

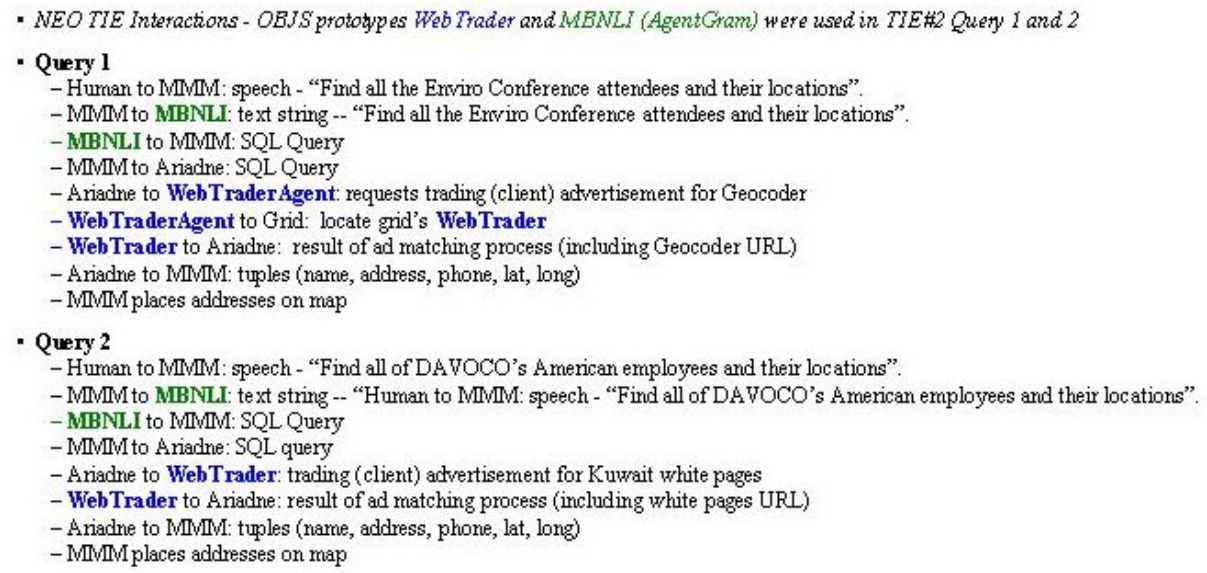


Figure 27: How MBNLI was used in the NEO TIE Scenario

The TIE required the following extensions to MBNLI:

- OAA-compatible MBNLI wrapper agent
- limited class inheritance capability (IS-A) and improvements to MBNLI grammar to project all join attributes for TIE joins, e.g. making "List the people and their addresses" equivalent to "List the people who have addresses -format including name, phone, address, latitude, longitude"

The following figure shows the NEO TIE MBNLI interface:

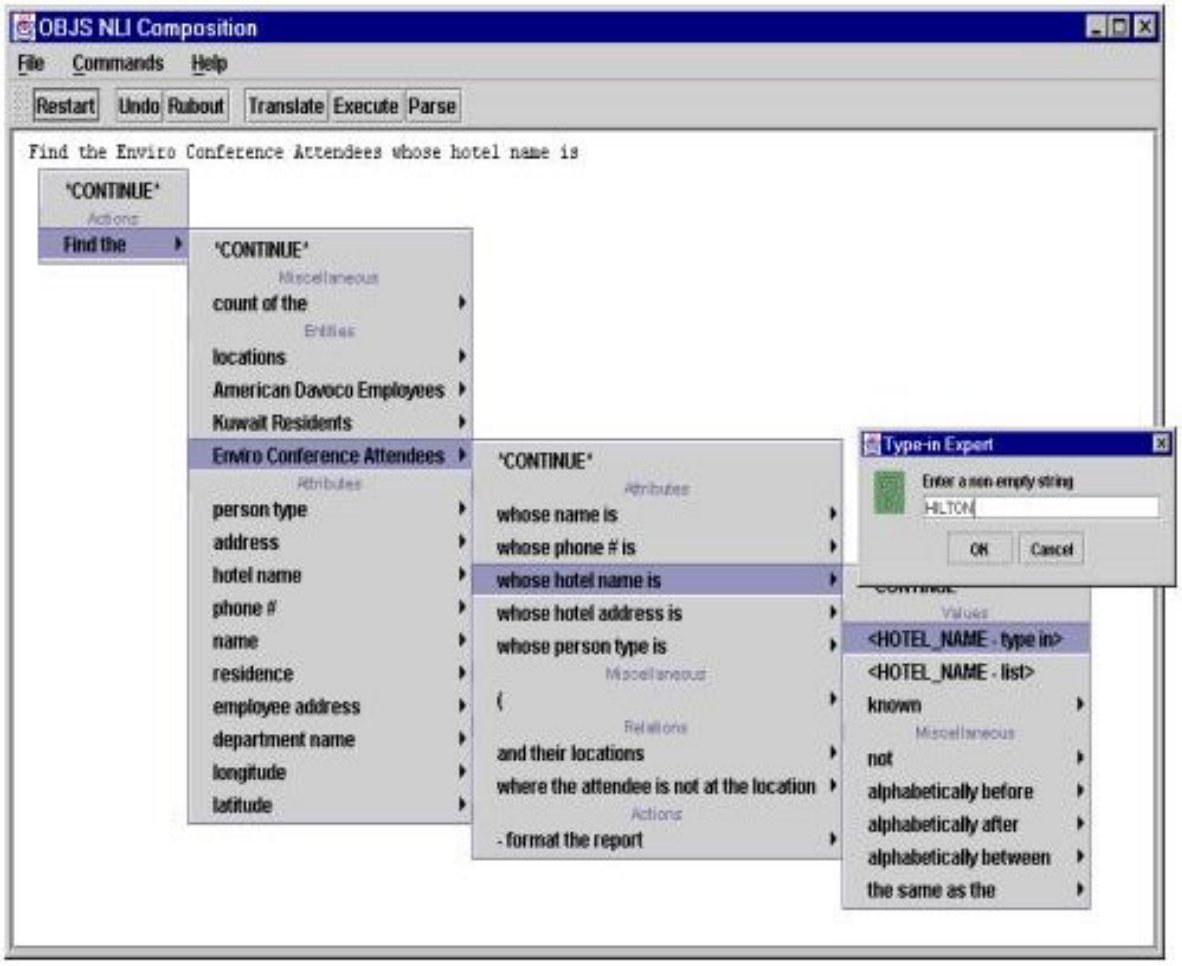


Figure 28: Specifying a NEO TIE query using MBNLI

3.4.2 CoAX TIE

In 2000 and 2001, we participated in the **CoAX TIE** [*http://www.aiai.ed.ac.uk/project/coax/demo/2001/*]aimed at demonstrating CoABS technology in a coalition scenario. Our work on AgentGram was featured in the Laki Safari Park Vignette described below. In this vignette, OBJS eGents (agents that communicate using email) send biosurveillance reports (e.g., location of elephants threatened by a planned UN firestorm in Safari Park Binni Wildlife vignette) to a DBMS. AgentGram was used to find the location of elephants near the planned UN firestorm. This was demoed at CoABS Workshop in Miami and Nashua. See **CoAX TIE avi** (.exe includes TechSmith TSCC Codec and viewer - 4.2MB - *http://www.objs.com/agility/tech-reports/0107-CoABS-Agility-Nashua/OBJS-CoAX-TIE-1024-768-256-TSCC.avi.exe*).

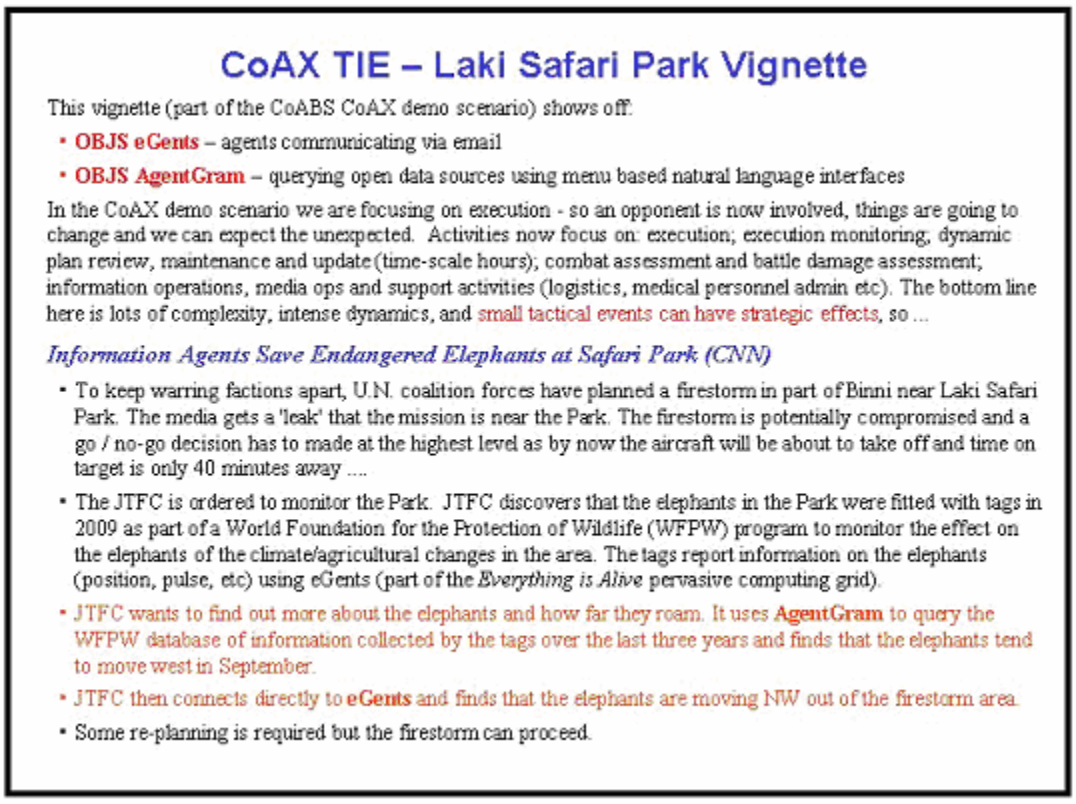


Figure 29: CoAX TIE Scenario

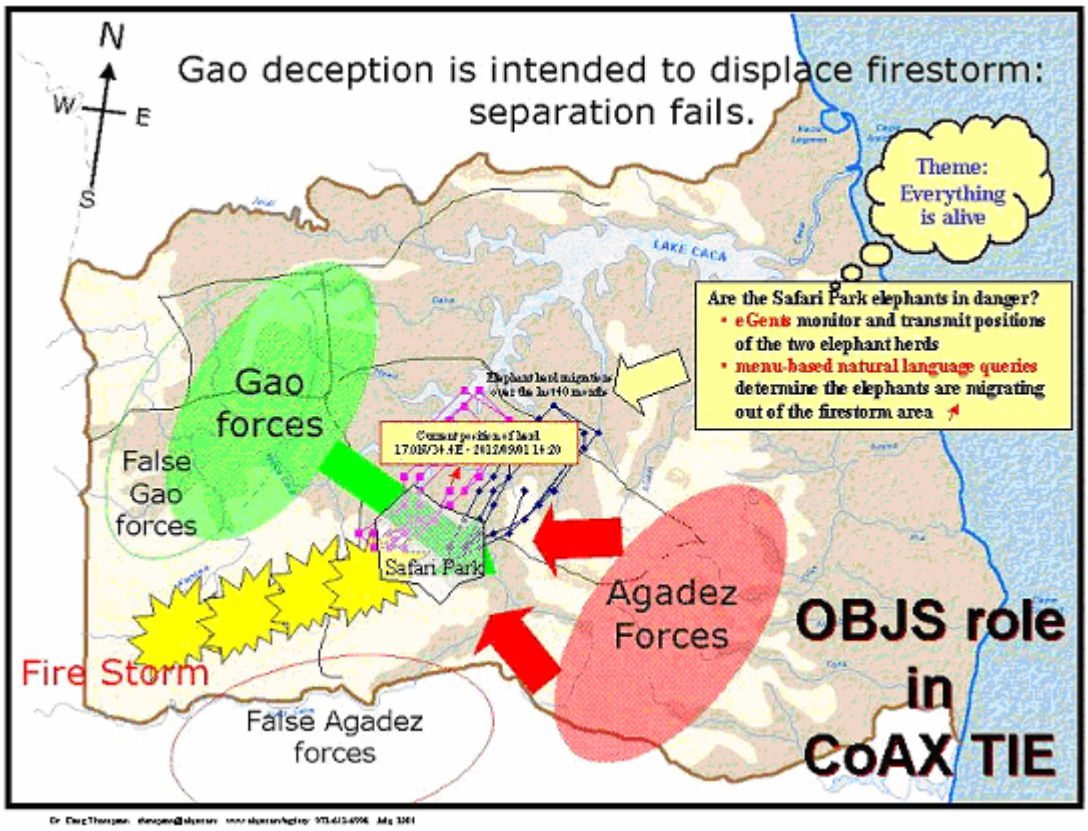


Figure 30: Are the Safari Park elephants in danger?

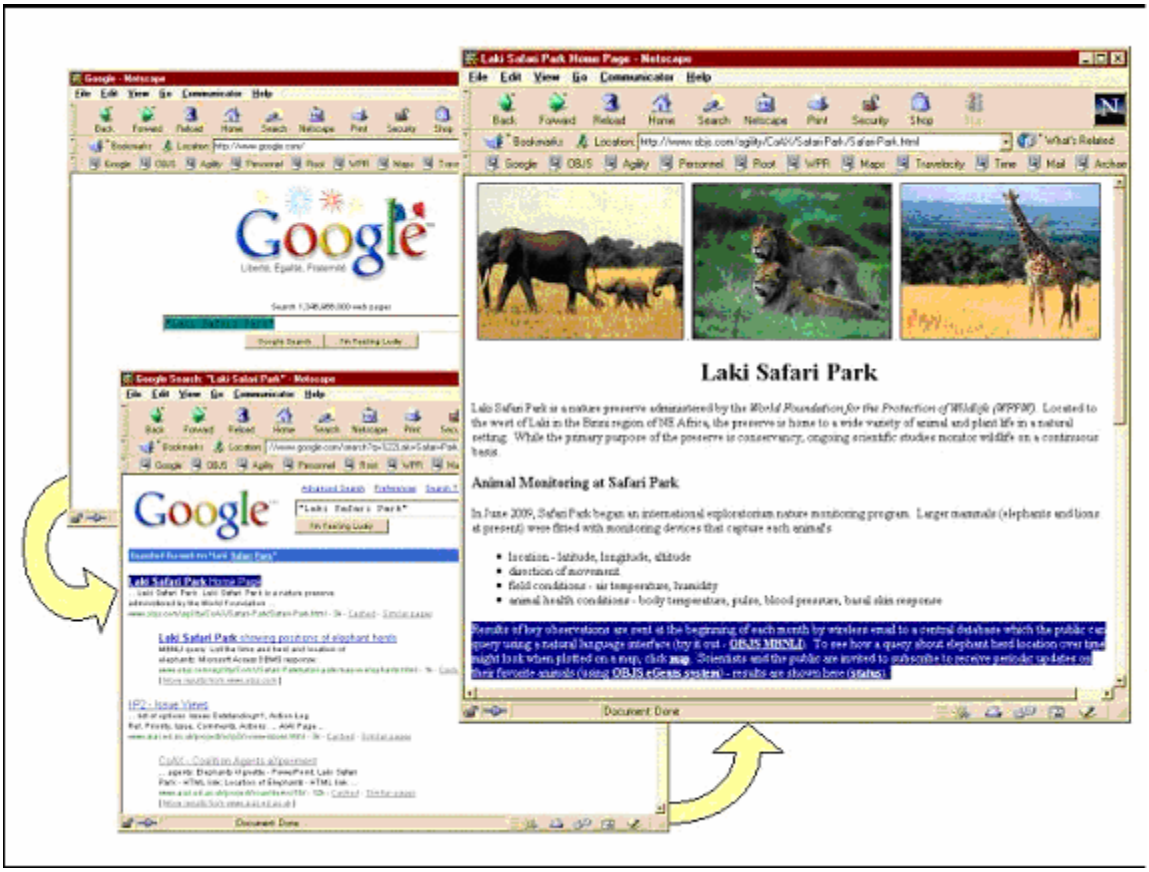


Figure 31: Finding our about Safari Park using the open Web

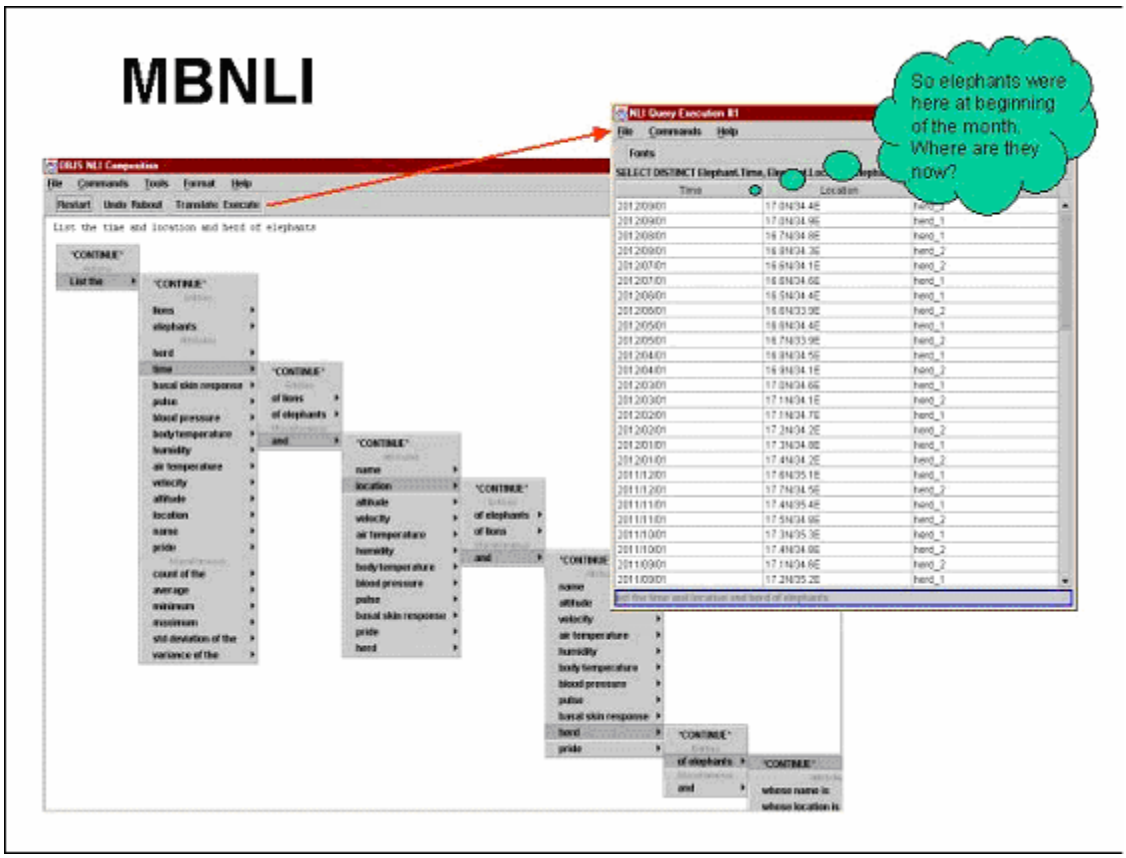


Figure 32: MBNLI Query to find recent elephant locations

3.5 Summary

Problem. Agent-based command and control systems offer real promise in flexibility and adaptability but it is not yet clear how they can be understood and controlled. It would be desirable if we could just talk to systems of agents - but how?

About Menu-based Natural Language Interfaces (MBNLI). Try typing or speaking to a system that has a conventional type-in or speech natural language interface (NLI). Most of your questions and commands will not be understood because they overshoot the capabilities of the NLI system or the underlying application it interfaces to. In addition, you won't really know about some things you could ask (map or statistical queries?) so your questions and commands will also undershoot the capabilities of the NLI system. Menu-based Natural Language Interface (MBNLI) technology uses standard NLI technology but in a menu-directed completion-based way to restrict the language and guide the user to just the capabilities of the NLI and underlying system. The technology fills a large unfilled niche in user interface design enabling unskilled users to make complex queries and commands.

Objective. The AgentGram project addresses human-to-agent communication via agents that understand constrained MBNLI languages. Humans can task and query one or more agents using complex but understandable commands. This technology can mix pervasively into all applications, both on the desktop and the Web, providing NLI capabilities to not just agents but also information sources, services, and generally any accessible resource.

Approach. Like other NL technology, MBNLI technology uses attribute grammars and a predictive parser. The difference is in using the grammars to predict legal sentence completions and displaying these using menus.

Progress. MBNLI, using a client-server parser farm architecture, supports **(a)** web-based access, **(b)** multiple users, **(c)** multiple simultaneous grammars, **(d)** speech control via Java Speech Markup Language, **(e)** the ability to generate MBNLI interfaces to DBMS system given the DBMS schema (and to extract a schema from a DBMS, then generate the MBNLI interface), **(f)** a partitioning of MBNLI into client and stateless server so a very thin client can be downloaded (with no installation) only requiring JavaScript (alternatively, there's an Java applets version), **(g)** interface descriptors on the "semantic" web, discoverable by traders or the grid, **(h)** techniques for composing MBNLI grammars and interfaces, **(i)** ported to IE, **(j)** patent application. The AgentGram prototype explores the idea of attaching NL wrappers to agents to make it possible for people to query or task them. Sub-grammars of individual agents can be traversed on-the-fly to construct commands or queries which span several agents. The web prototype shows how to make MBNLI technology pervasive on the web.

Demonstrations. (a) In the NEO TIE, the MBNLI component was agentized as an SRI OAA user interface agent (part of the SRI Multi-Modal Map) that translates natural language queries to SQL to be executed by an information access agent (ISI Ariadne). **(b)** In OBIS demos shown at the Science Fair, MBNLI was used to query the DBMS of Enviro Conference attendees and also to visualize the XML-based grid log. The Boston AgentGram demo finds MBNLI-enabled agents using WebTrader and shows how the sub-grammars of one or more agents can be traversed to produce commands and queries. **(c)** Our CoABS Grid demo illustrates one example of how our MBNLIGridAgent can be used to integrate MBNLI technology into the grid. In the demo the user is able to interact with the MBNLI web browser interface while the MBNLIGridAgent is able to provide current status regarding the interaction. **(d)** The CoAX TIE demo shows MBNLI used to extract data on elephants from a bio-surveillance data source at Safari Park in the Bimu fire zone area.

Technology Transition. A MBNLI agent was used in the Science Fair NEO application coupled to SRI OAA and ISI Ariadne, and modified for use with the CoAX TIE. Commercialization.

System Requirements. MBNLI web client requires Netscape browser; server requires WinNT, ODBC (Access, MySQL, Oracle), a cgi/php-capable web server.

Potential Directions. In the near term, we need to **(a)** demonstrate web-resident interface descriptors **(b)** interface AgentGram to agents and data sources in-the-wild (on the open Web and in DoD).

Figure 33: Summary

Chapter 4

WebTrader Prototype: Agent Discovery

4.1 Objective

The objective of the WebTrader project is to develop *WebTrader Query Tool (aka WebTrader)*, a trader/matchmaker/yellow pages that can scale to WANs and permit anyone on the Web to advertise a resource (e.g., agent, service, data source, search engines) that anyone else can discover. The approach to scaling is to represent advertisements in XML, store them on Web pages, let existing already pervasive and industrial strength search engines index these pages, then WebTrader's engine accesses one or more search engines, locates pages with advertisements, matches the ads against the request, and returns the matching advertisements.

Figure 34: Overview of WebTrader

One interesting application of WebTrader Query Tool is *DeepSearch*, a recursive/federated implementation of WebTrader that acts like a normal search engine but locates search engines at local sites and recursively searches these. Many web search engines only index top-level pages leaving the other half of the web unindexed except by these local search engines.

Specific scientific and engineering subgoals were:

- represent Internet resource advertisements for services, agents, data sources, search engines, traders. (This opens the door to the ontology problem that is the focus of the DARPA DAML program.)
- locate these advertisements via WebTrader which itself piggybacks on one or more Internet search engines that have indexed the resource advertisements
- demonstrate trader federation and rebinding
- re-engineer client-server WebTrader so the client is an applet that can download quickly to most browsers
- explore how to locate and mine local domain search engines as a way to broaden and deepen Web searching
- demonstrate WebTrader as a CoABS grid agent

- demonstrate WebTrader in scenarios of interest to DoD

WebTrader alone is not an agent system, but it is a generalized component capability of most agent systems, providing matchmaking. The interesting aspect of WebTrader is its scalability to the Web. Seen in this light, WebTrader can be viewed as a standalone agent grid enabling component, but one that operates in open WAN environments, is compatible with not only agent but also object and ontology technologies, and has no downloading barrier to widespread deployment.

4.2 Technical Accomplishments

4.2.1 Architecture

The basic architecture of WebTrader is fairly simple. Essentially, WebTrader is a new kind of specialized meta search engine that wraps other search engines to return typed advertisements from the open Web. As shown in the figure below, any page on the Web (1) can be annotated by anyone with an XML WebTrader advertisement (2) - see example **Trading Advertisement DTD** [*http://www.objs.com/agility/final/prototype-WebTrader/docs/trading-advertisement.dtd*]. These pages are indexed (3) by ordinary Web search engines. When a Query (client advertisement) (4) is presented (5) to WebTrader's engine (6), it accesses known Web search engines (7), they find matching web pages in the usual way (8) and return then (9). A WebTrader matching algorithm (10-12) finds candidate advertisements, scores them, and returns a series of responses in rated order (13-14).

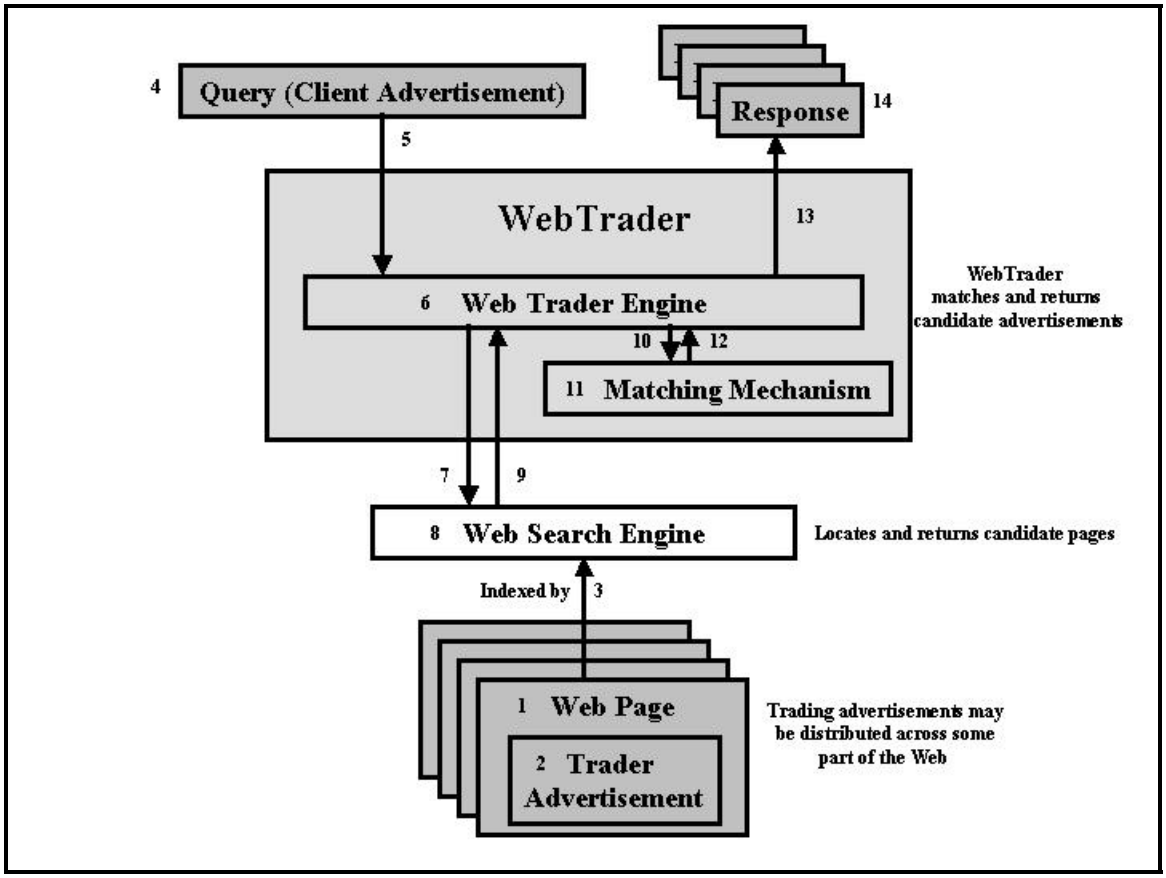


Figure 35: WebTrader Architecture

4.2.2 Internet Resource Advertisements, Trader Federation and Rebinding

Our Salt Lake City demonstration of the WebTrader (January 1999) illustrated:

- XML-based resource advertisements
- service binding and rebinding
- trader federation

A significant evolution in the design of the WebTrader ads occurred along the way, as we realized we could rework advertisements so that resource description in ads could be independently defined, allowing anyone to create an ad for anything they cared to define or use an existing definition of, from CORBA IDL components to bicycles. This change also affected ad matching and returned results. The revised TradingAd DTD now defines <!ELEMENT *resource* ANY>, allowing resources to be any XML document, from reconnaissance reports to parts inventories, from Java RMI components to Oracle databases. This successfully decoupled resource definition, basically an ontology question, from the WebTrader design, a major step forward.

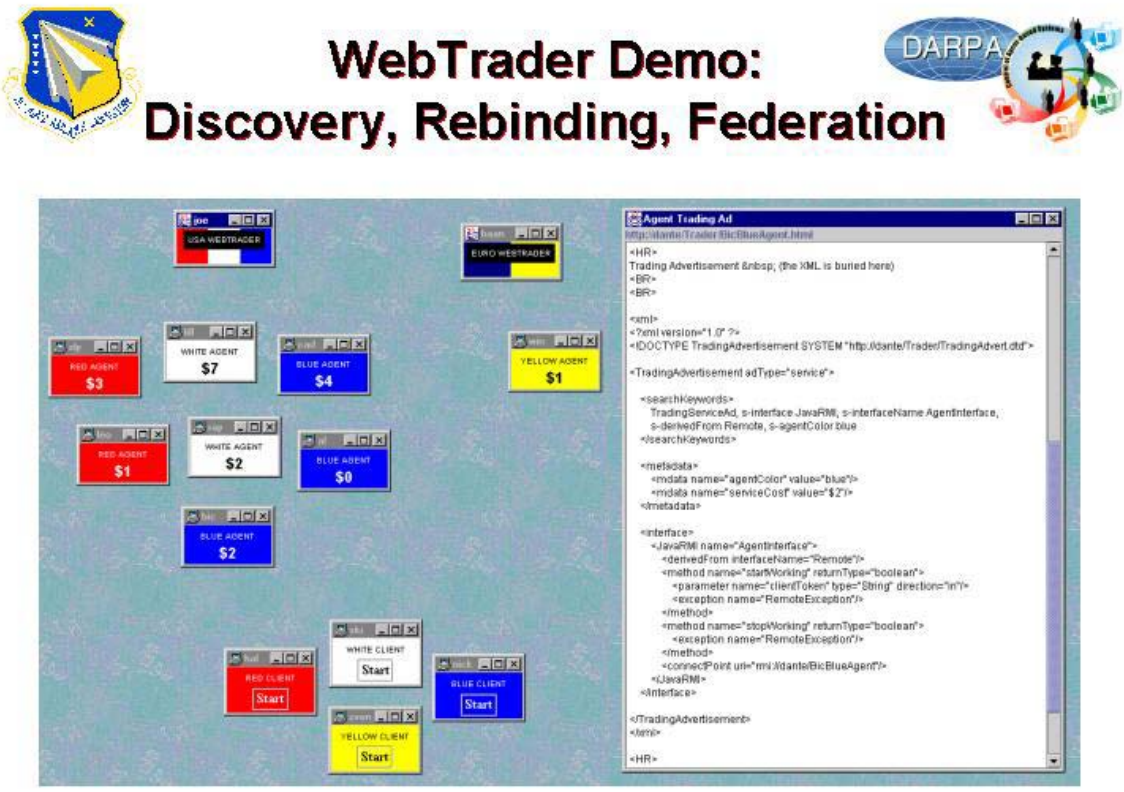


Figure 36: Discovery, Rebinding, and Federation

We also made progress on service binding and rebinding and trader federation. In the Salt Lake City demo, trading ads exist for a variety of components (e.g., service agents, clients, and WebTraders). Metadata including color and cost is included into the ads. A Blue client, for example, asks the USA WebTrader to locate a Blue agents implementing a particular interface and with zero cost, if possible. One is found and bound, and the client makes use of the agent. When the agent unexpectedly dies, the WebTrader is consulted again by the client, in case the "state of the world" has changed. It gets back a new list of agents, sorts them by cost, and goes down the list trying each agent until it finds one that works. If the WebTrader fails to respond, the client can fall back on its cached list of previously found agents. When the Yellow client asks the USA WebTrader for Yellow agents, the WebTrader's initial search turns up none, as it only consults a domain that indexes ads of Red, White, or Blue agents. However, it does find an ad for a WebTrader that knows about Yellow agents, and so passes on the original client query to the Euro WebTrader, which finds a Yellow agent, passes it back to the USA WebTrader, which passes it back to the client, which then uses it to connect to the agent. At the right in the figure is shown a service advertisement in XML. For more detail, see **WebTrader Demo Script** [*http://www.objs.com/agility/tech-reports/9905-WebTrader-Demo-Script.html*].

4.2.3 WebTrader Query Tool

Our status in Salt Lake City was, we had a better understanding of both webtrading and also the deep search problem but separate implementations. The next and main phase of the WebTrader project was to develop WebTrader Query Tool, a re-designed and re-implemented next generation WebTrader/DeepSearch capability that combined and generalized all functionality in a single prototype. This next generation WebTrader was engineered so that the front-end works as an applet with good performance on most browsers (tested on Internet Explorer and Netscape) and the backend is a servlet.

In a demonstration from the CoABS Boston Workshop (August 2000), WebTrader Query Tool was coupled with a modified Java App Builder (a visual programming tool), extended with WebTrader drag-and-drop capability. In the demo, the user creates a *want ad* called ColorButtonsSpec.xml, drags it to WebTrader Query Tool, runs a query which returns pages containing ads for colored buttons, then selected buttons are dragged to the WebTrader App Builder and connected together to build a simple application.

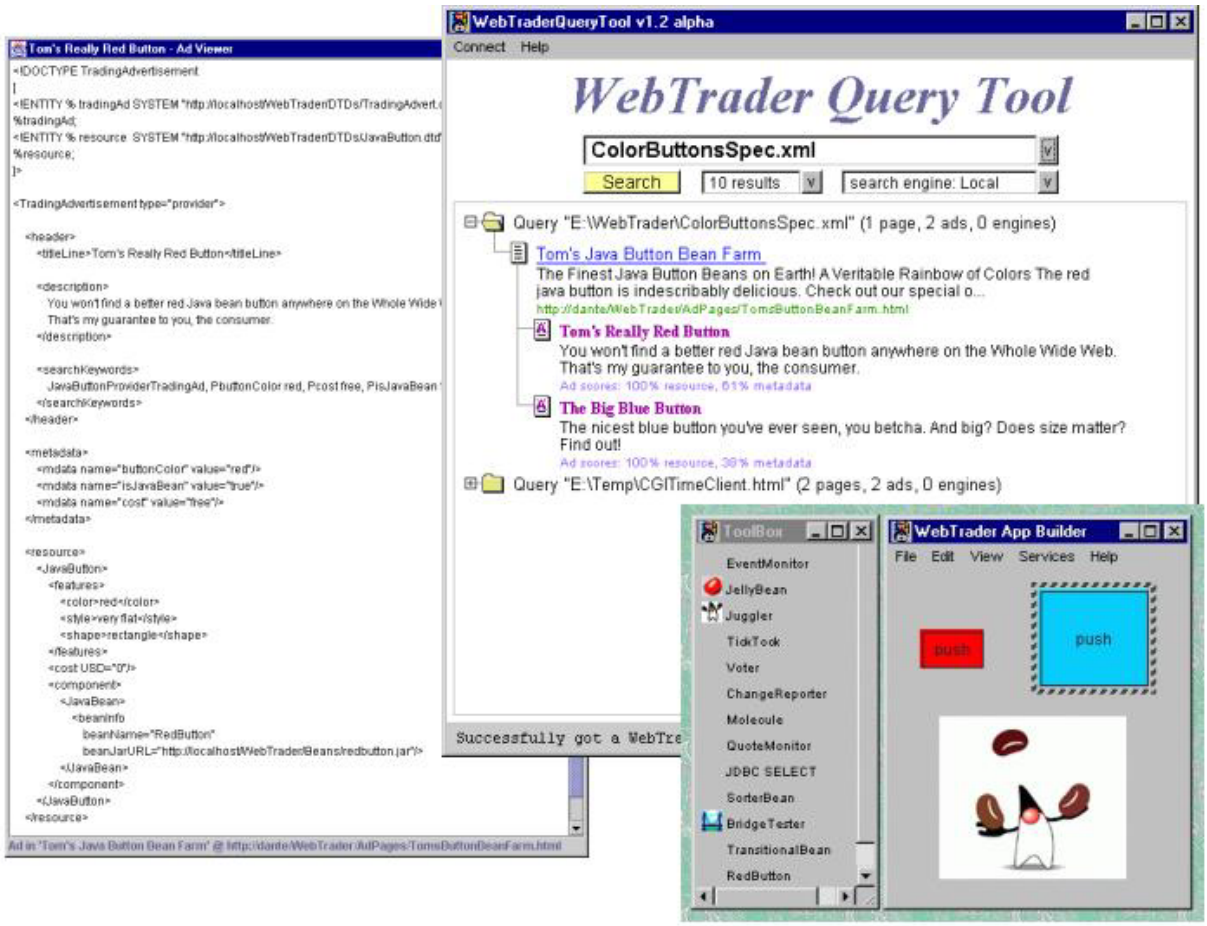


Figure 37: WebTrader Query Tool

WebTrader Query Tool is a generalization, redesign, and reimplementation of previous work in numerous ways -- the more significant changes are listed below. Almost all code was rewritten from the Salt Lake City implementation.

- **advertisements** - We redesigned the XML advertisements. The new design included <!ELEMENT *resource* ANY> as described above. We redesigned the WebTrader's JavaRMI DTD (now called JavaRMIObject), modified its CGI DTD, and modified all the example ad web pages to reflect these changes. We also created a new ad type (via a DTD) called SiteSearch to advertise local search engines. We located several local site search engines that indexed content having to do with agents and manually created ads for them. Later, we developed heuristics for discovering unadvertised local search engines on webpages. If found, we include these as a sub-result of that page and add it to the ad repository - this kind of discovery is a general approach to populating ad repositories.
- **applet-servlet architecture** - We separating searching from the display the results, and implemented an applet-servlet architecture. We reimplementing the client-side GUI as an

applet, around 30KB in Java 1.02 for widest portability, about half for the Tree GUI component. The tree component is used to record query histories, multiple live searches, canceling and deleting queries, adding increments to a search (e.g., top 20, 50, or 100 search results. The applet also supports a polling capability to handle the deep results as well as the top-level ones, scrollbars, and a concern for efficiency, to reduce the number of messages between the servlet and applet, so that web-sized scaling of the search service can be realistically achieved.

- **inputs** - WebTrader Query Tool can take either search terms or want ads as input. We made WebTrader Java Web Start-able, ie. one can download and run DeepQ as a desktop application with one click. When run on JVM 1.2 or later, such as via Java Web Start (JWS) or the Java Plugin, then dragging the WebTrader results over to the Web Trader App Builder (WTAB) is automatically enabled.
- **target search engines -** We developed a modular search engine results parser and specialized it to Webinator, Google and later to Open Directory. This kind of thing is an ongoing effort, and not just with Google, as SEs all seem to tweak their output syntax from time to time. We redesigned and implemented the match information returned to clients. Originally we expected to return W3C DOM parse trees of the XML ad resource to clients as part of this information and build convenience routines for clients to pick out desired information from the resource (such as the URL of a Java RMI object to connect to get the service advertised in the ad). However, the DOM objects generated by Sun's XML parser are not (directly) serializable and thus not able to be sent through the RMI link the WebTrader uses between its client and server parts. So, instead we return the raw XML of the resource and use SAX-based XML parsing tools on the client side to accomplish the same thing, a better design since we are sending less information and SAX is lighter weight than DOM. Later, we developed an algorithm to accurately parse the results of foreign search engines encountered on the fly (or fail safely).
- **matchmaking algorithm** - We modified the WebTrader metadata matching algorithm to more successfully match metadata dynamically created from user inputted search keywords. Also, we developed a generic scoring model.
- **federation** - We developed a new WebTrader federation design. One insight from the new design of the WebTrader federation scheme is that a WebTrader does not have to wait to determine that it cannot find any or enough matches to satisfy a client; at any point it comes across an ad for another WebTrader that looks promising it can propagate its query to it, and have all the results dynamically merged (sorted) on the client end. In fact, there is probably not any other practical way - what WebTrader has time for secondary WebTraders (and so on) to build up a list of results - all results must be streamed dynamically to be timely.
- **performance** - We implemented a new result polling capability. Now all search engines are equally controllable from the applet, instead of special consideration given to the top level engines in the old design. Now, the GUI will not keep up constant connection to servlet, but will rather poll for results.
- **security** - We modified the applet GUI to better work as a JWS app for the situation where there is no web page that the applet is embedded in.

4.2.4 Gridifying WebTrader

The CoABS grid is a JINI-based implementation of an agent interoperability platform developed by GITI, the DARPA CoABS program integration contractor. It is an important, on-going experiment in agent system interoperability. As described elsewhere, we contributed architectural ideas to the grid. But in addition, we developed three standalone agent components (eGents, WebTrader, and AgentGram) that can play a role as (stand alone or connected) grid components or services. As part of the Agility WebTrader project, we developed the grid-relevant capabilities described below. *At the same time, we note that WebTrader is itself a potentially pervasive stand-alone grid capability that can supply matchmaking for agent, object or ontology systems.*

For the Science Fair (October 1999), we gridified the WebTrader Agent (WTA). Previously, in NEO TIE #2 (see description below), the WebTraderAgent (WTA) was used as a WAN matchmaker to provide SRI OAA and USC/ISI Ariadne the ability to dynamically extend its information sources based on user queries it received. For the Science Fair, we modified the WTA to query the Grid for any WebTraders registered there. If none is found (or timeout), then the WTA automatically falls back to its local embedded WebTrader. This essentially extends CoABS grid JINI lookup service to be a WAN lookup service. The implementation involved creating the WebTrader Grid Service (WTGS), a Java program that registers a WebTrader on the CoABS Grid and logs its actions, along with a new version of WebTrader Agent (WTA) that lists WebTraders found on the Grid. We worked with Hank Seebeck and Adam Wenchel (both GITI) to test WTA on the grid. For the Boston CoABS Workshop (August 2000), we upgraded WebTrader Query Tool to use the grid in the same way as shown in the diagram below.

We also created a launch page for a CoABS 7x24 grid accessible version of DeepQ. WTA was maintained on the CoABS 7x24 grid for approximately a year.

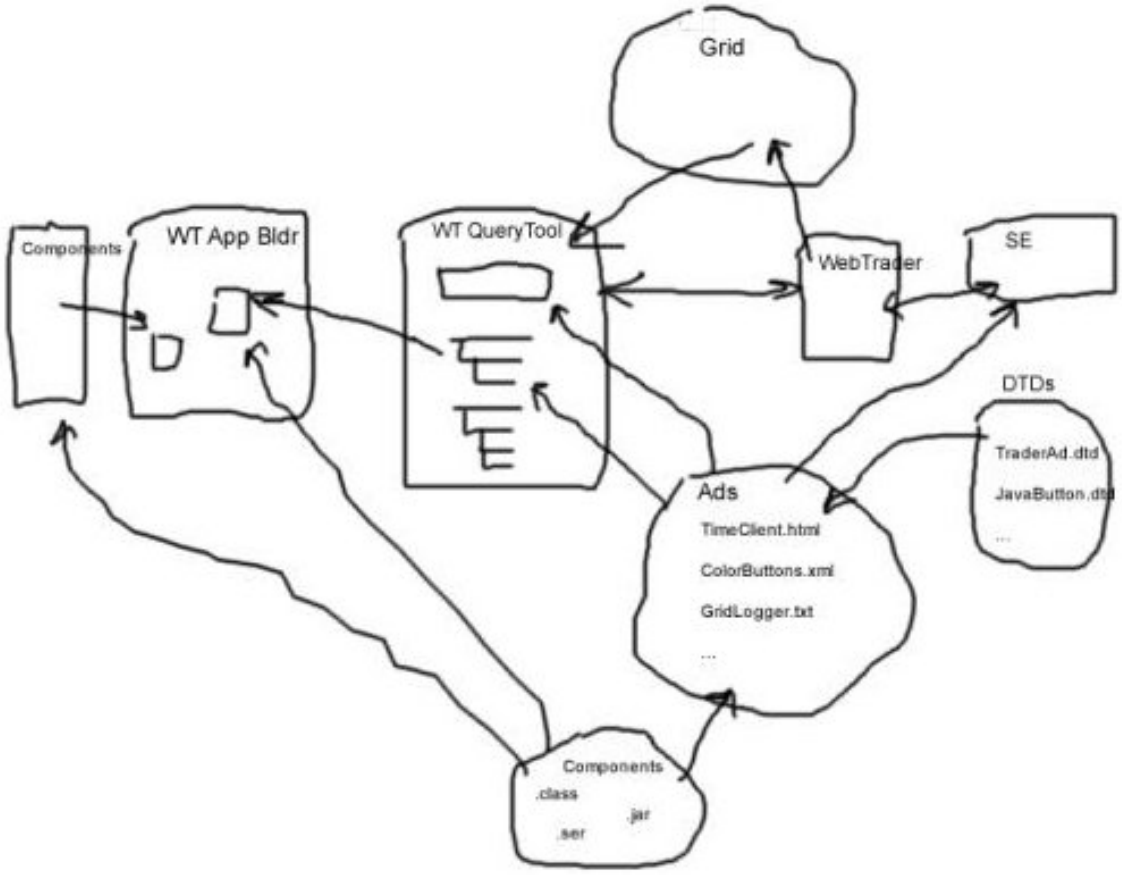


Figure 38: WebTrader on the Grid

4.3 Technology Transition

In 1999, we applied WebTrader in the DARPA CoABS Non-combatant Evacuation Order (NEO) Technology Integration Experiments (TIE). The NEO TIE involved an urban rescue effort and served to organize many CoABS program activities in the first year of the CoABS program. The thrust was on agent interoperability and rapid assembly of heterogeneous agent systems to solve problems. Lessons learned and components from this exercise where later incorporated into the CoABS grid. TIE #2 involves the scenario **Find Civilians, Get Them to Embassy** [*http://www.objs.com/agility/tech-reports/9809-TIE2-draft.html*], and the interaction scenario below shows not only what information is required but also the role of various CoABS subsystems: OBJS WebTrader, OBJS AgentGram/MBNLI, ISI Ariadne (which extracts information from Web pages), and SRI Open Agent Architecture (OAA, which enables inter-agent communication). Steve Minton (ISI) coordinated the TIE. For the NEO TIE, we developed WebTraderAgent, an OAA wrapper for a WebTrader. This effectively added a Web-wide trader to OAA. In the demo, when a new service or data source is needed, the

WebTraderAgent is consulted to locate a WebTrader which can then be consulted to locate advertisements for matching services. In addition to building the OAA wrapper, this involved porting WebTraderAgent to JDK 1.2 (Java 2), creating a stripped-down security policy for WebTraderAgent, and using an executable jar file to house the agent.

In March 2000, we presented work on **Agent Discovery Matchmaking** [*http://www.objs.com/isig/Agent-technology-white-paper-04.html#AgentDiscoveryMatchmaking*] to OMG, essentially, a sketch of a reference model for traders that subsumes WebTrader as well as conventional agent matchmakers and LAN traders like OMG's Trader.

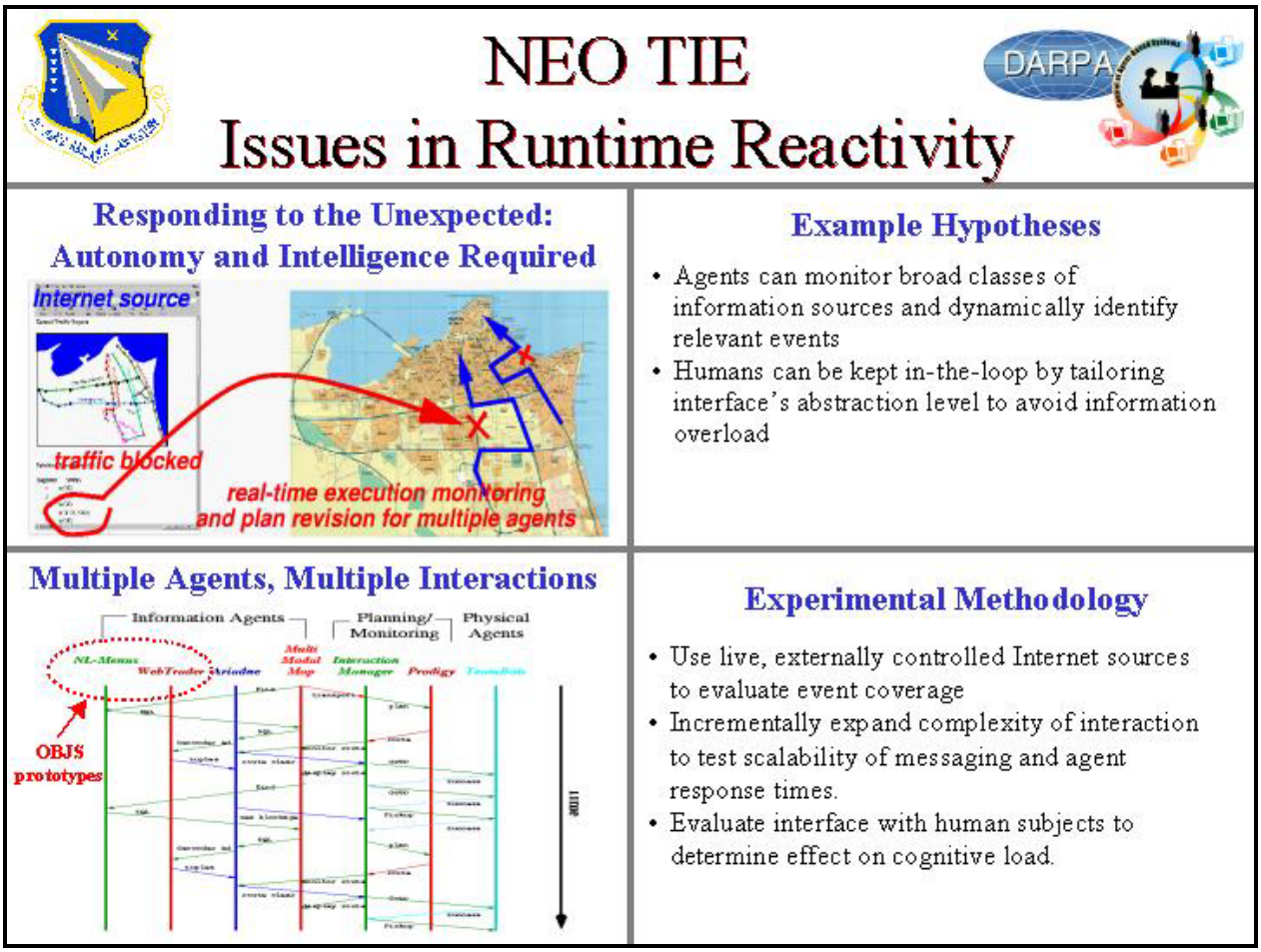


Figure 39: NEO TIE used WebTrader and MBNLI

- *NEO TIE Interactions - OBJS prototypes WebTrader and MBNLI (AgentGram) were used in TIE#2 Query 1 and 2*
- **Query 1**
  - Human to MIMM: speech - "Find all the Enviro Conference attendees and their locations".
  - MIMM to **MBNLI**: text string -- "Find all the Enviro Conference attendees and their locations".
  - **MBNLI** to MIMM: SQL Query
  - MIMM to Ariadne: SQL Query
  - Ariadne to **WebTraderAgent**: requests trading (client) advertisement for Geocoder
  - **WebTraderAgent** to Grid: locate grid's **WebTrader**
  - **WebTrader** to Ariadne: result of ad matching process (including Geocoder URL)
  - Ariadne to MIMM: tuples (name, address, phone, lat, long)
  - MIMM places addresses on map
- **Query 2**
  - Human to MIMM: speech - "Find all of DAVOCO's American employees and their locations".
  - MIMM to **MBNLI**: text string -- "Human to MIMM: speech - "Find all of DAVOCO's American employees and their locations".
  - **MBNLI** to MIMM: SQL Query
  - MIMM to Ariadne: SQL query
  - Ariadne to **WebTrader**: trading (client) advertisement for Kuwait white pages
  - **WebTrader** to Ariadne: result of ad matching process (including white pages URL)
  - Ariadne to MIMM: tuples (name, address, phone, lat, long)
  - MIMM places addresses on map

Figure 40: NEO TIE Vignette Scripts

4.4 Next Steps

The main puzzle preventing widespread use of WebTrader is the problem of populating the Web with advertisements. Since advertisements can be about anything, this is essentially the ontology problem. The entire DARPA DAML program as well as W3C RDF and semantic web communities are working this problem. Universal Description, Discovery, and Integration (UDDI) has recently been working to register businesses and their services. Finally, various Want Ad repositories have developed interesting proprietary ontologies. Since WebTrader's architecture is relatively independent of resource descriptions, the architecture can use any or several of these ontology representations. Both DAML and UDDI are examples of manually creating advertisements. An alternative is to automate this process (insofar as possible). A potential source of ads is the Web itself, namely, mining Web pages for ads of interest, especially in tractable domains. Alta Vista has done this with various easy-to-recognize data types like images and mpg's. It might not be hard to find certain other kinds of elements, for instance, banner ads. Similarly, our DeepSearch project has developed ad hoc recognizers for locating local search engines on Web pages. Of course, both manual and automated means of populating the ad space would be useful.

It is worth noting that the basic design of WebTrader appears to be that of an *open world* repository, that is, ads can come from any pages that search engines index. Interestingly, it is not a large change to convert WebTrader into a *closed world* search engine only containing ads that are inserted into its database (instead of SE index). This might make more sense for some kinds

of services, e.g. resume services or a company's web services, where the service provider might want control over who can advertise.

Although robust in several ways, the WebTrader implementation is still a prototype. More work is needed in several areas:

- We would like to explore type specific matchmaking so different matchmakers could be plugged into WebTrader on-the-fly depending on the type of advertisement. We ran into this requirement in the NEO TIE where it would have been useful to match the predicate, the arguments, and the variables in OAA solvables so only exact matches would be found
- We need more work on the algorithm for parsing results of search engines to extend the meta search capability.
- For many purposes, it makes more sense to index advertisements during the crawling phase rather than when pages containing ads are returned from search engines.
- We would like to remove the dependence on Thunderstone's Webinator which requires a live Internet connection (so it can check the online license at Thunderstone), does not always rank the results of OR queries correctly, does not support some of the query logic we need, and does not index XML pages. Open Directory ISearch is a potential alternative.
- We would like to use toolkits like WEBL and XWRAP Elite for web page content extraction as a way to locate SEs on web pages faster.

4.5 Summary

Problem. Most agent systems contain some sort of trader (matchmaker or yellow pages) that they use to locate agents by description. This provides late binding of agents to some service provider but does not scale outside of the closed agent system. Also, the trader itself is usually an integral part of agent system implementations and not available separately.

Objective. Build a scalable robust trader component architected for the global grid.

Approach. WebTrader is a scalable trader that locates advertisements (represented in XML) stored on web pages that have been indexed by search engines. Query results allow clients to connect to the most suitable service. Advertisement *types* include agents, components, data sources, search engines, MBNLI grammars, other traders, channels, and other types. WebTrader leverages industrial strength Web search engines so it is scalable, lightweight, portable, and robust. WebTrader is extensible -- recent changes to WebTrader support adding specialized matchers for ads of specific types. DeepSearch is an application of WebTrader that recursively searches other search engines and traders (using a federation design pattern). This makes specialized Web pages (not indexed by many search engines) accessible to DeepSearch. Also keeps a search history.

Demonstrations. (a) WebTrader was used in the Science Fair NEO application to locate data sources for the location of evacuees and to find a geocoder component. In the demo, ISI Ariadne uses SRI OAA to request WebTraderAgent to find these resources. WebTraderAgent consults the Grid to find a WebTrader service and then uses it to discover candidate resources which are returned to Ariadne via OAA. **(b)** Other demonstrations show trader federation, service rebinding, and WebTrader used to locate agents that use natural language wrappers. **(c)** The DeepSearch demonstration shows WebTrader locating ads for search engines on the web and recursively searching them, opening up parts of the web that are usually beyond the reach of general purpose crawlers. **(d)** WebTraderQueryTool is a recent hybrid of WebTrader and DeepSearch that adds matching and better supports application composition. Patent application.

Technology Transition. We led an effort of OMG Agent WG to review an Agent Resource Description and Discovery Service. We reviewed Agent UML. We are hardening WebTrader and DeepSearch getting ready for a public experiment and possible commercialization.

System Requirements: WebTrader Agent & WebTrader Grid Service require: MS Win NT 4.0 (pure Java programs), Apache (latest) web server, Webinator 2.5 search engine (from www.thunderstone.com) , Sun JDK 1.2.2, Sun XML TR-2 XML parser, GNU Make 3.75, tcsh. DeepSearch requires all of the above plus Netscape or IE, ActiveState Perl 5. Need connection to Internet (licensing quirk of local search engine used).

Potential Directions. WebTrader was put **on hold** as we are added additional resources to eGents. Things that need doing: **(a)** harden WebTrader and make the design more open; **(b)** grow WebTrader ad populations by making it easier to create WebTrader ads manually using a GUI; **(c)** try to harvest ads from the web to automate the process of ad synthesis (a web-size ontology question); **(d)** make matching ad type open so specialized search and ranking algorithms can be plugged in.

List of Acronyms and Abbreviations

ACK	acknowledgement
ACL	Agent Communication Language
AFRL	Air Force Research Laboratory
ALP	DARPA Advanced Logistics Program
CoABS	DARPA Control of Agent Based Systems Program
CoAX TIE	Coalition Agents eXperiment
CORBA	OMG Common Object Request Broker Architecture
DAML	DARPA Agent Markup Language Program
DARPA	Defense Advanced Projects Research Agency
DTD	XML Document Type Definition
FIPA	Foundation for Intelligent Physical Agents
GITI	Global Infotek, Inc.
IDL	OMG CORBA Interface Description Language
JBI TIE	AFRL Joint Battlespace Infosphere Program
KVM	K Virtual Machine (Java for devices)
MBNLI	Menu-based Natural Language Interface
MIATA TIE	Mixed-Initiative Agent Team Administration
NEO TIE	Non Combatant Evacuation Order
OBJS	Object Services and Consulting, Inc.
OMG	Object Management Group
PDA	Personal Digital Assistant
PSM	Personal Status Monitor
SIG	Special Interest Group
TIE	Technology Integration Experiment
UDDI	Universal Description, Discovery, and Integration
XML	Extensible Markup Language
W3C	World Wide Web Consortium
WAN	Wide Area Network